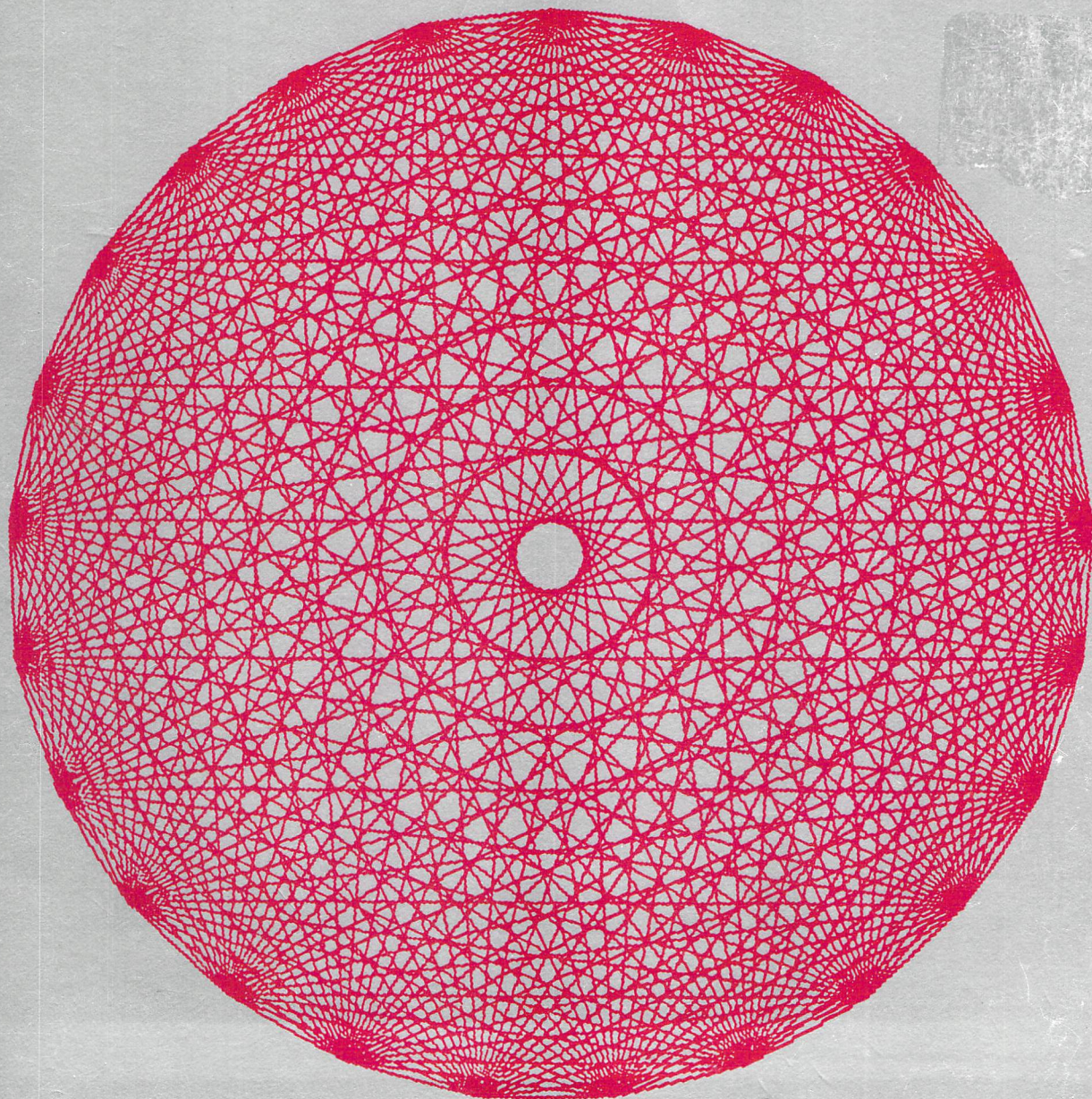


John Warner Davenport

Graphics for the Dot-Matrix Printer

How to Get Your Printer to Perform Miracles





Other books by John Warner Davenport:

Baseball's Pennant Races: A Graphic View

Baseball Graphics

How to Get
Your Printer
to Perform
Miracles

GRAPHICS for the DOT- MATRIX PRINTER

JOHN WARNER DAVENPORT



Computer Book Division
Simon & Schuster, Inc.
New York

Copyright © 1985 by John Warner Davenport
All rights reserved
including the right of reproduction
in whole or in part in any form
Published by
the Computer Book Division
Simon & Schuster, Inc.
Simon & Schuster Building
Rockefeller Center
1230 Avenue of the Americas
New York, New York 10020
SIMON AND SCHUSTER and colophon are registered trademarks of
Simon & Schuster, Inc.

Apple is a trademark of Apple Computer, Inc.
Macintosh is a trademark licensed to Apple Computer, Inc.
IBM and Selectric are trademarks of International Business Machines
Corporation.
Radio Shack and TRS-80 are trademarks of Tandy Corporation.
Microsoft is a trademark of Microsoft Corporation.
KoalaPad and Touch Pad are trademarks of Koala Technologies
Corporation.
Thunderscan is a trademark of Thunderware, Inc.
Epson is a trademark of Epson Corporation.
MX-80, FX-80, FX-100, RX-80, RX-100, and HX-20 are trademarks of
Epson America, Inc.
NEC is a trademark of NEC Corporation.
Prism is a trademark of Integral Data Systems, Inc.
PrintMate 150 and PrintMate 99 are trademarks of Micro Peripherals, Inc.
Laserphoto is a trademark of Associated Press.

The cover shows a printout made from the NETWORK program by
Delmer D. Hinrichs. This program was published in the May 1983
edition of *80 Micro*. Used by permission of Delmer D. Hinrichs
and *80 Micro*.

Designed by Irving Perkins Associates
Manufactured in the United States of America

1 2 3 4 5 6 7 8 9 10

Library of Congress Cataloging in Publication Data

Davenport, John Warner, 1931-
Graphics for the dot-matrix printer.

Includes index.

1. Computer graphics. 2. Printers (Data processing
systems) I. Title.

T385.D38 1985 001.64'43 85-11808
ISBN: 0-671-52338-4

FOR ARLENE AND LESLIE

Contents

CHAPTER 1	Do-It-Yourself Computer Graphics	11
	The Do-It-Yourself Approach	11
	Some Samples	13
	Topics Not Covered in This Book	19
	Hardware and Software	20
CHAPTER 2	A Primer on Printers	22
	How Microcomputers Produce Hard Copy	22
	Control Codes	32
	Finer Points	42
CHAPTER 3	Welcome to Dotland!	45
	Graphic-Mode Control Codes	48
	Precise Printhead Positioning	56
	Trouble Codes	68
CHAPTER 4	The Bit-Image Building Blocks	80
	The Binary Dot Codes	80
	Bit-Image Screen Dumps	98
CHAPTER 5	How to Draw with Dots	108
	Lines and Curves	108
	Mathematical Forms and Designs	122
	Drawing Pictures	133
CHAPTER 6	Your Own Graphics Catalogue	179
	How to Make Graphic Art and Drafting “Tapes”	179
	Screens, Tints, and Textures	190
	Symbols	204

CHAPTER 7	Your Own Alphabets	212
	How to Make Numbers and Letters	212
	Word-Spelling Programs	223
	Typography in BASIC	235
CHAPTER 8	Bar Graphs	260
	Horizontal Bar Graphs	260
	Vertical Bar Graphs	276
	Pictograms	300
	Program Conversions	304
CHAPTER 9	Line and Range Graphs	305
	Line Graphs	305
	Range Graphs	322
	Tandem Bar-and-Line Graphs	336
CHAPTER 10	. . . And More Graphs	338
	Scatterplots	338
	Ladder Graphs	356
	Pie Charts	360
CHAPTER 11	Diagrams, Maps, Tables, and Grables	368
	Diagrams with Block and Line-Graphic Characters	368
	Bit-Image Diagrams and Maps	380
	Tables	384
CHAPTER 12	Computer-Aided Graphics	399
	Rough Overviews	400
	Precision Plotting	402
	Manual Polishing	405
	Multigraphs	408
	Montages	415
	Printer Graphics in the Near Future	416
APPENDIX A	Data Codes for Two LINEDRAW-Type Programs	421
APPENDIX B	Code Sequences for Fourteen Custom Alphabets	428
APPENDIX C	Checklists for Converting TRS-80 Programs for IBM-Epson and Apple Printers	442

GRAPHICS for the DOT- MATRIX PRINTER

Do-It-Yourself-Computer Graphics

THE rushing stream of progress in microcomputer graphics is leaving behind a surprising amount of unfinished business. One of the underdeveloped areas of this field, far removed from the razzle-dazzle of color and motion in three dimensions, is printer graphics—the use of dot-matrix printers for high-resolution graphics. At the risk of setting the field back several years, this book will try to show the size and scope of possibilities that have been neglected in this area.

The Do-It-Yourself Approach

Throughout the book we will be dealing with the programming of drawings, graphs, diagrams, and tables for the printer, using BASIC programs and widely used personal-computing hardware. There are many graphic forms that can be produced by almost any microcomputer cabled to almost any printer that is dot-addressable—that is, any printer that allows you to program the firing of individual pins in its printhead.

This do-it-yourself approach need not involve any sacrifice of quality. Quite the contrary, in fact; nearly all printers have a higher resolution than the display screens of even the more expensive microcomputers. The Epson MX-80, FX-80, and RX-80 printers, for example, offer 960 dot positions in an eight-inch line and up to 216 dot positions per inch vertically. Even the monitors of the recent Tandy and Sperry

computers, which have 400 by 640 pixels (dots of light), can't match that. The resolution of the Epson printers, which is approached by a good many other brands in the three-hundred- to six-hundred-dollar price range of 1984, means that a high degree of detail and accuracy can be a routine matter.

This means that the do-it-yourselfer can get better printouts than those produced by some of the commercial software packages for graphics. Often the latter's approach is to create the desired graph or diagram on the computer screen and then do a "screen dump" to the printer, that is, print the contents of the screen. At best, this provides a close copy of what the screen displays, and it is precisely this that dooms the printout to a lower resolution than what the printer is capable of producing.

Often the result is an eyesore—lines and curves that have too obvious, jagged-looking "stair steps"; numbers along the axes that don't line up with hash marks or data points; primitive labeling of the axes that depends entirely on the orientation of the computer's and printer's built-in alphabetic characters; and other sins that would make the printout unacceptable for presentation in a respectable publication. For that matter, some of the software packages that don't depend on screen dumps commit many of the same sins. On the other hand, there are some on the market that produce excellent printed graphs, but even the quality of the best can be closely approached, if not equaled, by direct programming of the printer.

In nearly all of the programs presented in this book, no attempt is made to draw the graphic figure on the computer's screen. Instead, the computer is used to deliver the control and data codes required to generate high-resolution figures on the printer. The computer's screen still plays an important role in providing readouts of the codes being sent to the printer, displaying calculations, and producing other information for checking the programming during a printout. In addition, of course, the screen is used in the line-by-line programming itself.

In most of these programs the codes delivered to the printer are mainly directed to the firing of the individual pins of the printer's printhead, of which there are usually seven or eight that can be addressed at a particular instant in time.

Typically the pattern of the pin firing depends directly on the individual bits of an eight-bit byte (a dot code). This is why this approach is commonly called “bit-image,” “bit-graphics,” or “bit-mapping.” For an eight-bit printer such as the IBM (Epson) or the Apple Imagewriter, there are 256 different codes for 256 different patterns of dots produced by the firing of the pins—all possible combinations of the eight pins.

With this degree of control over the pins and the fineness of detail provided by the printhead pins, we should be able to draw almost anything—not just graphs, but complex diagrams, detailed maps, and photographlike pictures. This turns out to be true, but as we’ll see in later chapters, some drawings are much harder to do than others, in a reasonable period of time at least.

Some Samples

Harnessing the dots by programming in bit-image mode releases the user from many rigid printing formats and limits on the number of data values, variables, etc., which are imposed by a typical commercial graphics package. Most even limit you as to the type of graph you can produce—some software offerings costing over two hundred dollars limit you to pie charts and bar, line, and scatter graphs.

An investor wanting to track prices of stocks in the standard high-low-close fashion, for example, won’t be able to with such restriction. But with the do-it-yourself approach, the investor can design (or copy) a BASIC program that plots stock prices and volumes in a good imitation of *Wall Street Journal* style, as in figure 1-1. The more dedicated investor can add more measures to the plots, such as moving averages, relative strength, and the price/earnings ratio. Once the program is in good working form, plotting dozens or hundreds of stocks becomes a simple matter of typing more names and numbers into the computer’s datafiles. And if there are so many values that it takes a six-foot printout to display them, so be it.

Let’s switch to a radically different kind of graphics, such as the cartoon in figure 1-2. This drawing was created by a BASIC program which, outside of its DATA statements, is only eight lines long. With different items in the DATA state-

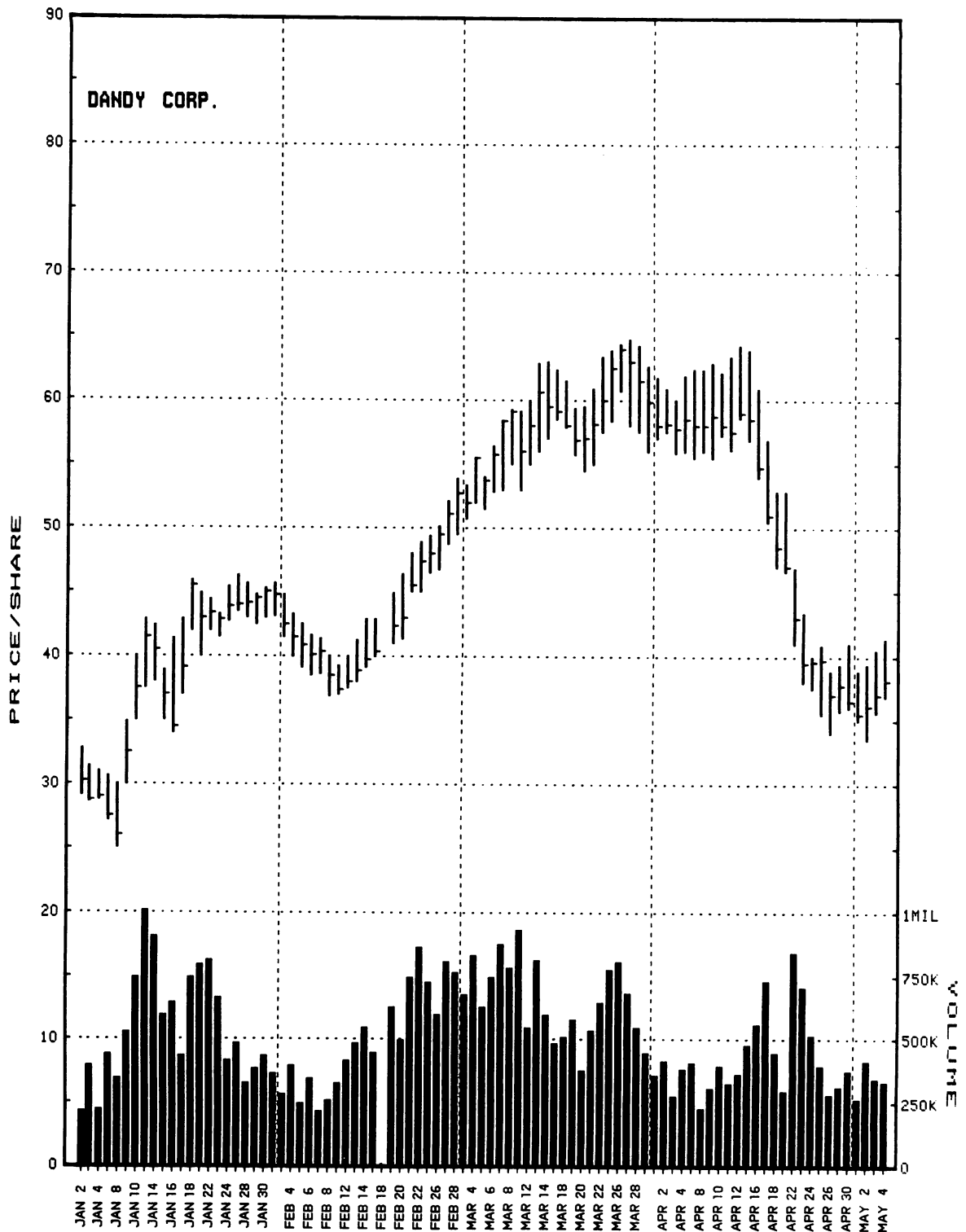


FIGURE 1-1. Daily changes in the price and trading volume of Dandy Corporation stock in the early part of 1994. (For details, see program 8-9, WALLSTOK, in chapter 8.)

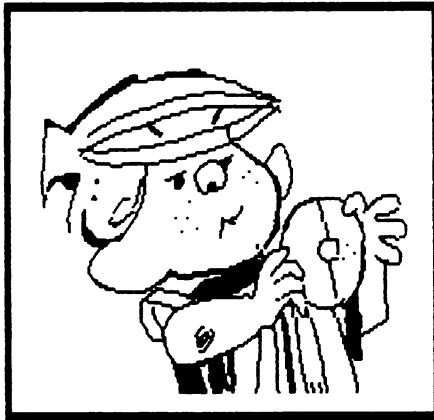


FIGURE 1-2. Cartoon drawn by the Radio Shack DMP-400 printer. (See program 5-10, LINEDRAW/TRS, in chapter 5.) (Dennis the Menace® used by permission of Hank Ketcham and copyright © by News Group Chicago, Inc.)

ment, the same program will produce other freehand drawings, simple or complex diagrams, or highly detailed maps (see programs 5-9 through 5-13). And with a few more program lines for calling subroutines, it will provide you drawings of nearly photographic quality.

For another example, a scientist may wish to plot a batch of bar graphs like the one in figure 1-3. Using the typical commercial program for a bar graph, there wouldn't be any problem displaying the means (arithmetic averages) of the data values, but if the researcher also wanted to indicate how the values were spread out from each mean, the program provides no way to add the customary thin vertical line representing a standard error or standard deviation at the top of each bar. This is an easy feature to program in BASIC, even when different shadings are used for the bars, as in our example. With a fresh ribbon in the printer, or with boldface graphic printing, homemade graphs done this way would meet journal quality in the eyes of most editors.

Or take the case of an illustrator for a newspaper that emphasizes graphics, such as *USA Today*. With the baseball season ending, the sports pages may be calling for a graphic overview of the major-league pennant races. A microcomputer driving a dot-matrix printer can plot the weekly standings of all teams in a division for the whole season as in figure 1-4. The lines connecting the letter symbols for each team can also be plotted by the printer, or the illustrator can choose to do these for one or all teams by hand, using drafting tapes, ink, or other graphic aids, to make the curves of certain teams stand out and to be sure of a more polished appearance.

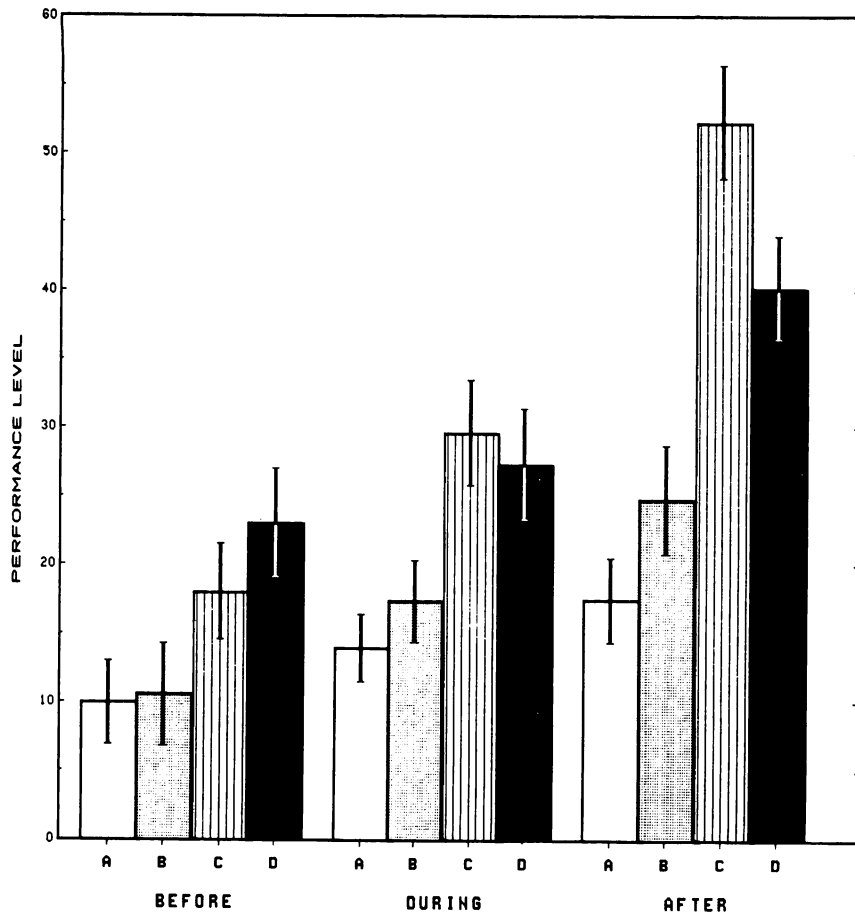
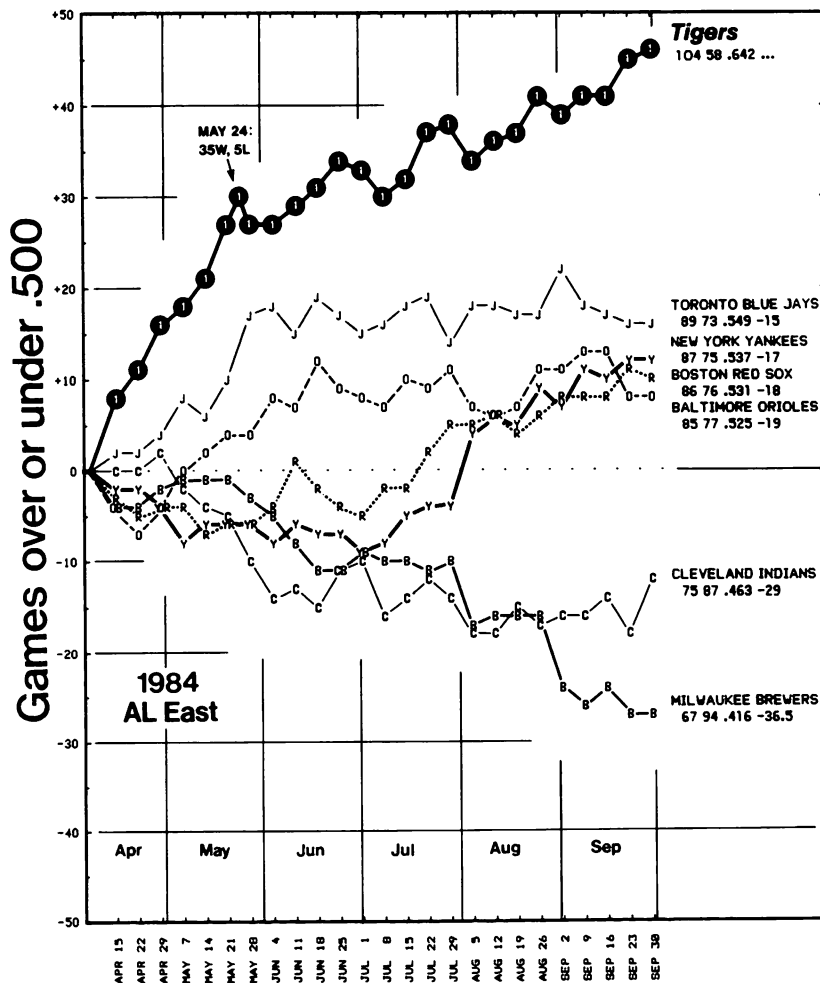


FIGURE 1-3. A bar graph in scientific journal style. (See program 8-7, MSEGGRAPH, in chapter 8.)

This idea of “tampering” with the printout can be carried much further—to the whole realm of computer-aided graphics. Those things that computers and printers are good at can be combined with those things that are best done by hand. Take the political map in figure 1-5, for example. This was produced by taking two copies of the map of the United States, which were printouts from a BASIC-programmed dot-matrix printer. One copy was printed white-on-black and the other printed regular black, both printouts on white paper. The regular black printout was identical to the one in figure 11-5 in chapter 11. The states ending up black (i.e., those won by the Republicans) in the finished map were cut out of the second copy with an X-Acto knife. Then on a light table the second copy was overlaid on the first, with state boundaries lined up as exactly as possible, making the cut-out states

emerge as black with white boundaries where they were adjacent to one another. Those states won by the Democrats were left white with black boundaries, and those won by the third party were covered with shading film on the second copy. Then a clear plastic overlay was taped over the first two sheets. Using dry-transfer ("rub-on") characters, the states were numbered and lettered, in white over the black-

FIGURE 1-4. A major-league baseball pennant race portrayed in a line graph. The graph was constructed by a program similar to MULTLING (program 9-2 in chapter 9) in combination with a second program for listing the teams and their won-lost records. Manual graphic finishing touches were added. (Reprinted with permission of Macmillan Publishing Company from *The Detroit Tigers* by George Sullivan and David Cataneo, graphics by John W. Davenport. Copyright © 1985 by Macmillan Publishing Company, a division of Macmillan, Inc.)



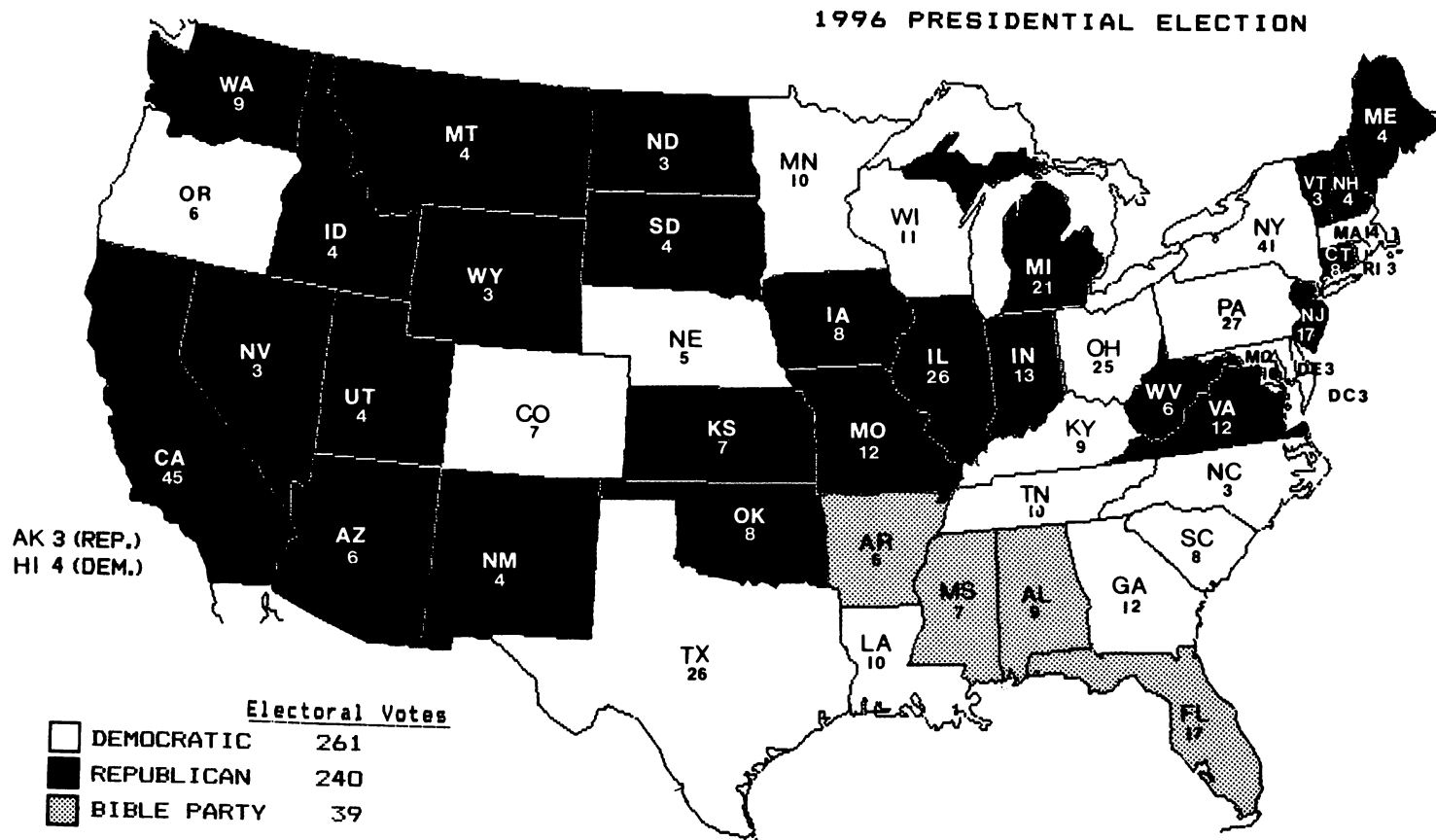


FIGURE 1-5. How the 1996 presidential election was thrown to the House of Representatives by a third party. Produced by overlays of printouts from a map-drawing program. (See chapters 11 and 12.)

ened states, and in black otherwise. Finally, the printer was used as a typesetting machine to produce the title and key of the map, which were applied to the clear overlay with ordinary glue.

These examples suggest the wide applications of directly programmed printer graphics. When you add this graphics capability to the several other roles served by printers—as text printers in word processing, as typesetting machines, as copying machines, etc.—the potential uses of the well-programmed dot-matrix printer run an even wider gamut. Not only scientists, businesspersons, and journalists, but meteorologists, cartographers, architects, self-publishers, graphic artists, and (one would think) computer-graphics buffs may find some useful material in later chapters.

Topics Not Covered in This Book

By now you can guess what this book is *not* about. Except for a short section about bit-image screen dumps in BASIC (chapter 4), almost nothing about screen graphics, especially color graphics, will be taken up. As a result, we will also not be concerned with animation (getting images to move on paper is not very easy!), or dynamic three-dimensional visualizations, or the various input devices—digitizers, X-pads, KoalaPads, and so forth—for graphic creations on the computer's display. And we will leave it to the hordes of computer magazines to review the commercial software, even though most of those packages provide printed copy on a printer or plotter.

Although we will concentrate on black-and-white products of printers (plotters are another whole book), we don't necessarily have to do completely without color in printer graphics. For one thing, some dot-matrix printers (e.g., the IDS Prism, the Transtar T315) print in several colors, depending on the position of a multicolored ribbon, which can be programmed. In addition, almost any printer can be fitted with a solid-color ribbon that is red, green, or brown, as you wish. And these colors need not be printed on white paper; for a Christmas design, why not try red printing on green paper?

Some clever work by Francis Kalinowski (in *80 Micro*, November 1983) shows several intricate four-color printouts

from an Epson MX-80 printer with Grafrax Plus. He printed each color using a different ribbon and started print runs for the different colors at the same point on the paper. His work also points the way toward generating unusual colors and shadings by combining colored inks in overlapping print runs.

Beyond these color possibilities, notice the color ink-jet printers coming into the microcomputer market in 1983 and 1984 at prices well under a thousand dollars. These printers use the same dot-matrix principle and the same kinds of control codes as the black-and-white printers except for a few additional commands for control of the colors.

Hardware and Software

Many of the programs in this book were originally designed for the Radio Shack TRS-80 Model LPVIII and Model DMP-400 printers, the latter being a widebody machine for large graphs, architectural plans, and tables up to fifteen inches in width. These were driven by the TRS-80 Models III and 4, and the TRS-80 Model 100 and Epson HX-20 briefcase portables.

Many of these TRS-80 programs were modified to run on the Epson FX-80 and related printers such as the IBM Graphics printer, and some programs were also written for the Apple Imagewriter. The IBM PC, Apple IIe (with Applesoft BASIC), and Apple Macintosh (with Microsoft BASIC) computers were used to test these programs, and the Tandy Model 2000 computer was also used to test some of the original programs for TRS-80 printers.

This has resulted in three main orientations of the book—Tandy, IBM, and Apple—with respect to both computers and printers. This plan should provide a good deal of help to users of other printers (Okidata, IDS, NEC, C. Itoh, Star, and Mannesmann Tally, among others), because printhead pin configurations and control codes within “families” of printers are fairly similar.

The main emphasis will necessarily be on programming for TRS-80 and TRS-80-related printers in later sections of the book, because these are the easiest to program when printer graphics becomes complex, and many of the drawings require a fifteen-inch printer such as the DMP-400. For the

benefit of IBM and Apple users, specific steps are presented for converting any program for TRS-80 printers in Microsoft BASIC to an IBM PC program (also Microsoft BASIC) for IBM-Epson printers, or to an Applesoft BASIC program for Apple printers. Examples are also given.

Computers with 64K or more of random-access memory and two disk drives would be most convenient for the programs in this book, but even a cassette-based computer within only 16K of memory will handle nearly all of the programs. A system that costs under a thousand dollars (including computer, printer, cassette recorder, and cables), then, can do the job even on some of the more complex programs for eight-inch printers, whose prices have recently fallen well below five hundred dollars in many cases. Anyone seriously interested in the full range of printer graphics, however, would be wise to spend another four hundred to six hundred dollars for a widebody printer. (Three of the five illustrations in this chapter were printed on a fifteen-inch printer.)

So far as computer capabilities are concerned, what matters nearly as much as memory size is how strong the computer's BASIC language is. With Microsoft BASIC in the versions found in the IBM PC, all the recent Tandy computers, and the Macintosh, you're in good shape. But if your computer lacks the `STRING$` function and a few other features of Microsoft that are particularly handy for printer graphics, it will be tougher going. An even greater handicap can result from the computer's inability to send eight-bit codes to a printer (this is most likely to be a problem with Apple II computers, as is the weakness of Applesoft BASIC). Ways of dealing with these problems will be presented.

This book assumes some working knowledge of BASIC, but only what an average person would pick up in ten to twenty hours starting from zero. Many programs do not go beyond such familiar BASIC terms as `PRINT`, `LPRINT`, `READ`, `DATA`, `INPUT`, `FOR`, `NEXT`, and `RND`. Some of the more advanced programs do involve setting up and sorting arrays, manipulating strings (`STR$`, `VAL`, concatenation), and reading information from datafiles, but these are accompanied by explanations. Almost no use will be made of machine language or other high-level languages such as `PASCAL`, `FORTRAN`, or `FORTH`.

2

A Primer on Printers

YOU might be thinking how nice it would be if we could plunge right into programming some graphic creations, but getting well acquainted with your printer first will prevent some disasters. This chapter will present some general information about dot-matrix printers and also cover the mostly nongraphic functions of these printers when they are used in text mode. (“Text mode” in this book will refer to what printer manuals variously call “text,” “data processing,” or “word processing” modes—in other words, any status outside of the bit-graphics mode.) The next chapter will deal with the graphic features and the control codes for all the things a printer can do in graphic mode. Both chapters are important for graphics, though, because we often have to intermix bit-image graphics with the printer’s built-in characters (in labeling graphs, for example) and with certain control functions that can only be reached outside of graphic mode.

How Microcomputers Produce Hard Copy

Material that is printed on paper is known in computerese as hard copy, as opposed to the “soft,” temporary, and often moving images and characters displayed on the computer’s screen. Quite a few devices are used to produce hard copy—letter-quality “daisy-wheel” and “thimble” printers, impact dot-matrix printers, thermal printers, ink-jet printers, laser

printers, electrosensitive printers, line-matrix printers, and plotters. This list can be boiled down to three categories: formed-character printers, dot-matrix printers, and plotters, since impact, thermal, ink-jet, electrosensitive, laser, and line-matrix printers all use a matrix of dots to print characters.

The formed-character printers are the ones most like typewriters. They have formed characters that produce an impression on the paper by striking through a ribbon much the way the formed characters on typewriters do. Also, the way in which characters are selected on a daisy-wheel or thimble printer (which are usually under the control of a word-processing program) is similar to the way characters on the “golf ball” of an IBM Selectric typewriter are selected by presses of the typewriter’s keys. Formed-character printers are excellent for producing high-quality printing of verbal and numerical material, but they are slower than impact dot-matrix printers and of almost no use in printing graphics.

At the other extreme, plotters are mainly designed to make graphic drawings and are rather limited in the quality of built-in characters they produce. By definition, plotters differ from printers in drawing with pens, usually several having different colored inks. Since the whole principle and programming procedures of plotters are so different from those of dot-matrix printers, there is very little in this book that applies to plotters, even though they can do most of the same things a printer can do in graphics.

A great deal of the material in this book, on the other hand, may apply to thermal, ink-jet, laser, and line-matrix printers. Some of the control codes may be different from those dealt with here, but all of these printers use the dot-matrix principle. Thermal printers make the dots with a printhead that has heated wires that create an impression on special temperature-sensitive paper. Electrosensitive printers also make use of special paper that blackens from tiny electrical charges. Ink-jet printers make dots by literally spraying miniature ink drops on ordinary bond paper or on higher-quality, polyester-based paper. You’ll probably be seeing a lot more of these in the future, because they are quiet, faster than impact printers, and in several cases offer a range of selectable colors. And at least three ink-jet models were priced under nine hundred dollars in early 1984. Electrosensitive printers, on the other hand, are well beyond the price range

EPSON FX-80

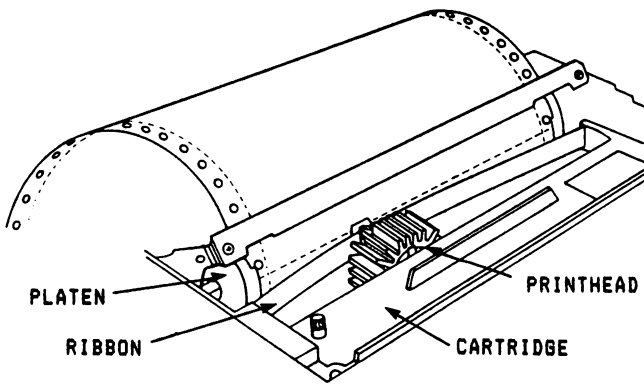
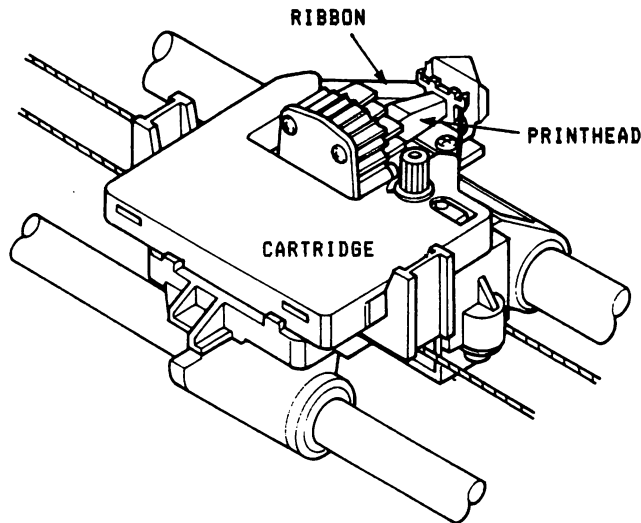


FIGURE 2-1. Printheads and surrounding features of the Epson FX-80 and Radio Shack DMP-400 dot-matrix printers. (Reprinted with permission of Epson America, Inc., and Tandy Corporation.)

RADIO SHACK DMP-400



of impact printers in the personal computing market, and laser printers have only recently begun to approach this price range.

Another innovation by Radio Shack that is also approaching that range is the line-matrix printer. The LMP-2150, initially offered for \$3,995 in early 1984, prints whole lines at a time and operates at a speed of up to 150 lines per minute. It uses the standard Radio Shack bit-image graphics codes, so much of the rest of this book should apply to its programming.

Certainly the most common type of dot-matrix printer is the impact type, in which moving pins (also called wires or needles) of the printhead strike an inked ribbon to make an

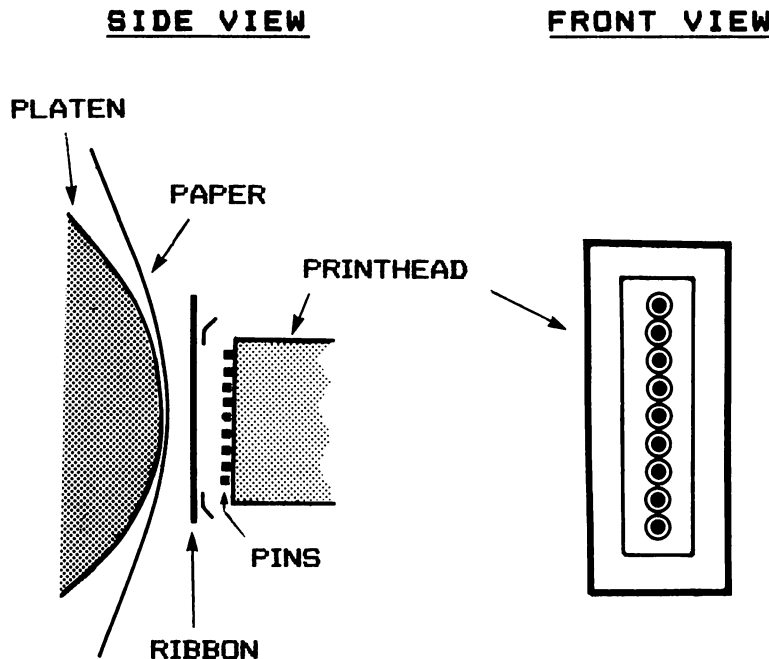
impression on paper. Figure 2-1 shows what the printheads of two widely used impact printers look like. You would have to be an Incredible Shrinking Person to see the printing as it happens because of the small spaces separating the printhead, ribbon, and paper.

Figure 2-2 lets you see a side view of the business end of a printhead and its pins in relation to the ribbon, paper, and platen (the cylinder carrying the paper). There is also a diagram of how the pins of the printhead look from the ribbon's point of view.

Three types of printers. We can sort most ordinary dot-matrix printers into three "families" on the basis of the number of printhead pins that are available for bit graphics and the top-to-bottom order of the pins. We'll call these the "IBM-Epson," "Apple," and "Tandy" (Radio Shack) families. Table 2-1 lists many other printers that are members of these families.

The first two of these clusters include all printers having

FIGURE 2-2. Side view (left) and ribbon's-eye view (right) of a typical printhead.



eight addressable pins, while the Tandy group consists of seven-pin printers. But the IBM-Epson and Apple families of printers differ in having their pins in opposite orders. Whereas the Apples have their top pin fired by the low bit (the 1 bit) of an eight-bit dot code, and their bottom pin fired by the high (128) bit, the arrangement is exactly reversed on the IBM-Epson printers—the high bit fires the top pin, and the low bit fires the bottom pin.

To those who are used to counting or numbering things from the top down, the IBM-Epson printers are upside down while the Apple is right side up, but this is the opposite of the way computing engineers view the world of bits and bytes. Printers in the Tandy family are like the Apples in

TABLE 2-1
Three Families of Dot-Matrix Printers

<p style="text-align: center;">IBM-EPSON</p> <div style="text-align: right;"> <p>● 128</p> <p>● 64</p> <p>● 32</p> <p>● 16</p> <p>● 8</p> <p>● 4</p> <p>● 2</p> <p>● 1</p> </div>	<p style="text-align: center;">APPLE</p> <div style="text-align: right;"> <p>● 1</p> <p>● 2</p> <p>● 4</p> <p>● 8</p> <p>● 16</p> <p>● 32</p> <p>● 64</p> <p>● 128</p> </div>	<p style="text-align: center;">TANDY</p> <div style="text-align: right;"> <p>● 1</p> <p>● 2</p> <p>● 4</p> <p>● 8</p> <p>● 16</p> <p>● 32</p> <p>● 64</p> </div>
<p>IBM Graphics Printer</p> <p style="text-align: center;">~</p> <p>Epson FX-80, FX-100</p> <p>Epson RX-80, RX-100</p> <p>Epson MX-80, MX-100</p> <p style="text-align: center;">~</p> <p>Star Gemini 10, 15</p> <p style="text-align: center;">~</p> <p>Mannesmann Tally MT160</p> <p>Mannesmann Tally Spirit</p> <p style="text-align: center;">~</p> <p>Inforunner Riteman</p> <p style="text-align: center;">~</p> <p>Panasonic KX-P1090</p>	<p>Apple Imagewriter</p> <p>Apple Dot Matrix</p> <p style="text-align: center;">~</p> <p>C. Itoh Prowriter I</p> <p>C. Itoh Prowriter II</p> <p style="text-align: center;">~</p> <p>NEC 8023A</p> <p style="text-align: center;">~</p> <p>Anadex DP-9000A, 9500A</p> <p>Transtar T315</p> <p style="text-align: center;">~</p> <p>Canon A-1200</p> <p style="text-align: center;">~</p> <p>Personal Micro DMP-85</p>	<p>TRS-80 LPVIII</p> <p>TRS-80 DMP-110, -120</p> <p>TRS-80 DMP-200</p> <p>TRS-80 DMP-400, -420, -500</p> <p>TRS-80 CGP-220</p> <p style="text-align: center;">~</p> <p>Okidata Microline 82, 93</p> <p style="text-align: center;">~</p> <p>IDS Prism 80, 132</p>

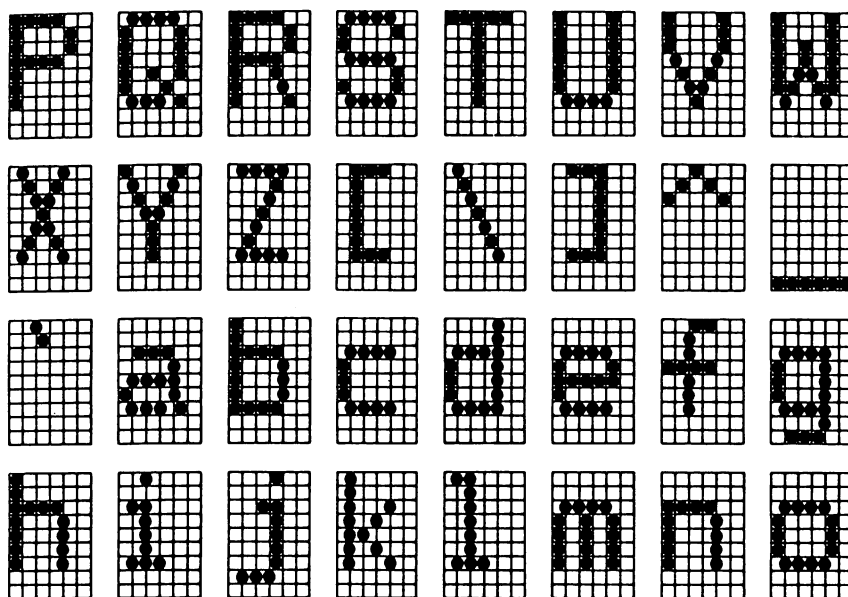


FIGURE 2-3. How upper- and lowercase characters are formed on the Epson FX-80 printer.

having their pins numbered “right side up,” that is, “low bit up.”

If you find this hard to follow, look at the numbers (1, 2, 4, 8, 16, 32, 64, and 128, the zeroth through seventh powers of 2 in the binary system) beside the printhead pins in table 2-1. For further help, peek ahead at the first three figures in chapter 4, which show all 256 dot patterns for both the IBM-Epson and Apple Imagewriter printers and the 128 dot patterns for the Tandy-family printers. Our three-family scheme, by the way, leaves out a fourth group of printers, which includes the Micro Peripherals Printmates 99 and 150G and the recent Anadex and Datasouth brands, having only six pins available for graphics. Also left out are the near-letter-quality dot-matrix printers having twenty-four pins in their printheads, such as the Radio Shack DMP-2100, Toshiba P1350, and Epson LQ-1500, and those having eighteen pins, such as the Radio Shack DMP-430.

Figure 2-3 indicates how characters are made out of dot patterns by dot-matrix printers. These samples are from a printer having nine pins, which is enough to produce lowercase letters having true descenders (the “legs” in g, p, q, and y) and to provide for underlining as well. Each character is

made from a matrix that is six dot-columns wide, but (except for the underline) the last column is blank and is not used for anything except spacing between characters.

It isn't accurate to call this a six-by-nine matrix, though. If you look closely at the characters in figure 2-3, you'll see that many of the dots are not entirely within the dot columns but instead are halfway between two columns. These are intermediate "half-dot-column" positions that the printer can use in forming the built-in characters, but which, unfortunately, are not available to the BASIC programmer. This amounts to a nine-by-nine matrix for the character plus a two-by-nine matrix of blanks for spacing, but with the horizontal positions more densely packed than the vertical dot positions. Obviously the quality of the printed characters is much better with these intermediate positions than would be the case if the printer were unable to use them.

Printing samples from the heads of our three families are shown in figure 2-4. You will see that all three produce good but not true letter-quality printing, and that there are differences among them. The Epson printers have an italic character set, which the IBM Graphics, Imagewriter, and Radio Shack DMP-400 printers do not. (The more recent DMP-430 in the Tandy line does, however.) The DMP printers offer a "correspondence" font that is closer to letter quality, but the Epson and Apple printers do almost as well in bold or emphasized printing. There are also differences in the number of "pitches" (characters per inch, or CPI); all three printers provide standard pica (10 CPI), elite (12 CPI), condensed (16.7 CPI), and elongated (5 CPI) pitches, but the Imagewriter offers two intermediate pitches (13.5 CPI and 15 CPI).

Of more interest to graphics buffs are the block and line-graphics characters. Thirty-one of these can be seen in figure 2-4 in the DMP-400 print sample, right after the lowercase alphabet. Although they are not available on the Epson FX-80, there are a few line-graphics characters on the MX-80, and a thirty-two-character set on the Epson RX-80. The IBM Matrix and Graphic printers have more extensive (sixty-four-character) sets than the Tandy DMP-series and RX-80 printers have. (This and the resulting lack of italic characters represent the main differences between the IBM and Epson printers, both of which are manufactured by Epson America, Inc.). The Imagewriter also lacks these built-in graphics

EPSON FX-80

SINGLE-STRIKE PICA	<i>SINGLE-STRIKE PICA</i>
SINGLE-STRIKE ELITE	<i>SINGLE-STRIKE ELITE</i>
SINGLE-STRIKE COMPRESSED	<i>SINGLE-STRIKE COMPRESSED</i>
SINGLE-STRIKE EMPHASIZED PICA	<i>SINGLE-STRIKE EMPHASIZED PICA</i>
DOUBLE-STRIKE PICA	<i>DOUBLE-STRIKE PICA</i>
DOUBLE-STRIKE ELITE	<i>DOUBLE-STRIKE ELITE</i>
DOUBLE-STRIKE COMPRESSED	<i>DOUBLE-STRIKE COMPRESSED</i>
DOUBLE-STRIKE EMPHASIZED PICA	<i>DOUBLE-STRIKE EMPHASIZED PICA</i>
SNGL-STRK EXPANDED PICA	
SNGL-STRK EXPANDED ELITE	
SNGL-STRIKE EXP. COMPRESSED	
SNGL-STRIKE EMPH. EXP. PICA	
DBL-STRIKE EXP. PICA	
DBL-STRIKE EXP. ELITE	
DBL-STRIKE EXPANDED COMPRESSED	
DBL-STRK EMPH. EXP. PICA	

APPLE IMAGEWRITER

```

0123456789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuv
123456789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvw
23456789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvw
3456789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
456789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
56789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz(
6789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz(l
789:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{l)
89:;<=>?@ABCDEFGHIJKLMN O PQRSTU VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{l)~

```

TRS-80 DMP-400

```

NORMAL PICA
NORMAL ELITE
NORMAL CONDENSED
BOLD PICA
BOLD ELITE
BOLD CONDENSED
NON-BOLD ELONGATED PICA
NON-BOLD ELONGATED ELITE
NON-BOLD ELONGATED CONDENSED
(BOLD AND ELONGATED CANNOT BE COMBINED)

ABCDEF GHIJ KLMNOP QRSTUV WXYZ1234567890<>?:;I; /
abcdefghijklmnopqrstuvw xyz!@#%&'()*_+-=,.
  " ' , . : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```

FIGURE 2-4. Printing samples from the heads of three printer families.

characters, but a close relative in the Apple family, the NEC 8023A printer, has an excellent set of fifty-six straight-line, curved-line, and block characters (shown in figure 11-1 in chapter 11).

BASIC printing commands. Like a microcomputer, a dot-matrix printer has a microprocessor and both read-only memory (ROM) and random-access memory (RAM) provisions. And just as the computer has a section of memory space for printing characters on its display, the printer has ROM sections for printing characters and interpreting control codes that it receives. Over much of the range of ordinary character codes in text mode, the printer's characters will faithfully match the characters appearing on the computer's screen. With most systems, for example, `PRINT CHR$(48)` will produce a capital H on the screen and `LPRINT CHR$(48)` will produce a capital H on the printer.

`LPRINT` is in fact the most common printing command for most combinations of computers and printers. The `LPRINT` command operates like the `PRINT` command in several ways. Just as `PRINT "A"` puts a letter A on the screen, `LPRINT "A"` will result in a hard-copy print of the letter A. `LPRINT A` will produce a print of the value of variable A, as with `PRINT A` on the screen. In fact, the identical operation of `PRINT` and `LPRINT` extends to computations (`LPRINT A/B-C`), concatenation of strings (`LPRINT A$+B$`), printing in spaced print zones (`LPRINT A,B,C`), formatted printing (`LPRINT USING` commands), and blank line feeds (`LPRINT` alone). And like `LIST`, the command `LLIST` gives you a line-by-line printout of a BASIC program.

Screen and printer outputs are similar, but not identical, for the `PRINT TAB(N)` and `LPRINT TAB(N)` commands. On a fifteen-inch printer `LPRINT TAB(N)` commands when N is above 80 may behave in strange ways; the `TAB` function is mainly designed for screen output. Suppressing line feeds by placing a trailing semicolon at the end of a BASIC statement also falls into the "similar but not identical" category. Near the far right end of a print line on a fifteen-inch printer, at least, a line feed may occur even if you try to prevent one with a trailing semicolon.

There is one type of `PRINT` command that has no corre-

sponding LPRINT command. That is the PRINT@ (or PRINT AT, or PRINT preceded by LOCATE) command. This type of command refers only to the computer's screen. The only way to locate the starting of printing in a similar way on the printer's page is to combine horizontal positioning with line feeds.

Some computers, the IBM PC being one of them, can use PRINT# as well as LPRINT for sending codes to a printer. The # in PRINT#, however, must be followed by a number that refers to a computer buffer, and that buffer must be numbered in a preceding OPEN statement, such as **OPEN "LPT1:" AS #1** ("LPT1:" refers to "line printer 1"). After the OPEN statement (and usually a WIDTH statement as well) the full print command takes the form **PRINT#1, CHR\$(N)** instead of **LPRINT CHR\$(N)**. PRINT#1, USING can be used instead of LPRINT USING, and in general the PRINT# command can do all the things that LPRINT can, and more. We'll see in the next chapter that PRINT# solves some problems with unwanted carriage returns and line feeds when you are programming the PC for bit graphics.

Some other computers, such as the Epson HX-20, do not use LPRINT at all for delivering codes to an external printer. Instead, they use a communications (COM0) channel, which has to be opened with an OPEN command along with the PRINT# command. The HX-20 uses LPRINT only for its built-in miniature printer, and communicates with an external printer with the statements **OPEN "O", #1, "COM0:"** and **PRINT#1, CHR\$(N)**.

Still another form of PRINT# in place of LPRINT shows up with the TRS-80 Color Computer. In this case, PRINT#2 must be used to send codes to a printer.

The Apple II computers have never used LPRINT in Applesoft BASIC. Instead, they use PRINT for both the screen and the printer. If the PRINT command is preceded by PR#0, the output will go to the screen; if preceded by PR#1, to the printer. In later examples, however, we'll see that it is usually necessary to substitute the POKE command for PRINT in Applesoft BASIC. The Macintosh, on the other hand, has Microsoft BASIC as standard software, and that makes it equivalent to the IBM PC in offering a choice between LPRINT and PRINT#.

Control Codes

Every dot-matrix printer has a set of control codes for selecting pitches, boldface printing, line feeds, carriage returns, and other nongraphic functions. As if it weren't already bad enough that printers differ in their pin arrangements, here we find not only differences among the three families, but some large differences within families. And yet it is not completely accurate to say that every printer has a different set of control codes.

TABLE 2-2
Control Codes in Text Mode for the Three Printer Families

	IBM-EPSON	APPLE	TANDY
LINE FEED (ACTIVATE)	CHR\$(10)	CHR\$(10)	CHR\$(10)
HORIZONTAL TAB (ACTIVATE)	CHR\$(9)	CHR\$(9)	(NONE)
FORM FEED	CHR\$(12)	CHR\$(12)	CHR\$(12)
CARRIAGE RETURN	CHR\$(13)	CHR\$(13)	CHR\$(13)
BACKSPACE	CHR\$(8)	CHR\$(8)	CHR\$(8);CHR\$(N)
DIRECT POSITIONING	(NONE)	CHR\$(27)"Fnnnn" (nnnn=0000-9999)	CHR\$(27);CHR\$(16);CHR\$(N1);CHR\$(N2)
HORIZONTAL TAB SET	CHR\$(27)"D"CHR\$(N1)...CHR\$(NK) CHR\$(8) (1 to 32 tab positions)	CHR\$(27)"*CHR\$(N1)...CHR\$(NK) (cleared by CHR\$(27)CHR\$(8))	(NONE)
VERTICAL TAB SET	CHR\$(27)"B"CHR\$(N1)...CHR\$(NK) CHR\$(8) (1 to 16 tab positions)	CHR\$(31)CHR\$(N) (N=66 to 78)	(NONE)
SET PICA (10-CPI) PITCH	(Default)	CHR\$(27)"M"	CHR\$(27);CHR\$(19)
SET ELITE (12-CPI) PITCH	ON--CHR\$(27)"M" OFF--CHR\$(27)"P"	CHR\$(27)"E"	CHR\$(27);CHR\$(23)
SET CONDENSED (16.7-CPI)	CHR\$(15) or CHR\$(27)CHR\$(15)	CHR\$(27)"Q" (13.4- and 15-CPI also available)	CHR\$(27);CHR\$(28)
EXPANDED PRINT -- ON EXPANDED PRINT -- OFF	CHR\$(14) or CHR\$(27)"W1" CHR\$(28) or CHR\$(27)"W8"	CHR\$(14) CHR\$(15)	CHR\$(27);CHR\$(14) CHR\$(27);CHR\$(15)
PROPORTIONAL PITCH	ON--CHR\$(27)"p1" OFF--CHR\$(27)"p0"	CHR\$(27)"P"	CHR\$(27);CHR\$(17)
ITALIC FONT	ON--CHR\$(27)"A" OFF--CHR\$(27)"S"	(NONE)	(NONE)
UNDERLINE -- ON UNDERLINE -- OFF	CHR\$(27)"-1" CHR\$(27)"-0"	CHR\$(27)"X" CHR\$(27)"Y"	CHR\$(15) CHR\$(14)
BOLDFACE PRINTING -- ON BOLDFACE PRINTING -- OFF	CHR\$(27)"E" or CHR\$(27)"G" CHR\$(27)"F" or CHR\$(27)"H"	CHR\$(27)"!" CHR\$(27)"**"	CHR\$(27);CHR\$(31) CHR\$(27);CHR\$(32)
PROPORTIONAL SPACING	(NONE)	CHR\$(27)"sn" (n=0 to 9)	CHR\$(27);CHR\$(N) (N=1 to 9)
LEFT MARGIN SET	CHR\$(27)"1"CHR\$(N)	CHR\$(27)"Lnnn" (nnn=000 to 999)	CHR\$(27);CHR\$(16);CHR\$(N1);CHR\$(N2)
REPEAT PRINTING	(NONE)	CHR\$(27)"Rnnn" (nnn=000 to 999)	CHR\$(28);CHR\$(N)
MASTER RESET	CHR\$(27)"@"	(NONE)	(NONE)
LINE FEED SET -- 1/6" LINE FEED SET -- 1/8" LINE FEED SET -- 1/12" REVERSE LINE FEED SET	CHR\$(27)"2" CHR\$(27)"0" CHR\$(27)"A"CHR\$(6) CHR\$(27)"J"CHR\$(N)	CHR\$(27)"A" CHR\$(27)"B" (NONE) CHR\$(27)"r"	CHR\$(27);CHR\$(54) (REPEAT CHR\$(27);CHR\$(58) 9 TIMES) CHR\$(27);CHR\$(28) CHR\$(27);CHR\$(38)

For some sample differences, look at table 2-2. This table presents the control codes of the “family heads” for all the important nongraphic commands. In order for these control codes to do their job, they have to be sent to the printer by LPRINT, PRINT, PRINT#, or whatever a computer uses as its primary line-printer command for printing characters in regular text mode.

For some commands only a single code is necessary to execute the command. For example, CHR\$(13) is the almost universal code for a carriage return, CHR\$(8) is widely used for backspacing, and CHR\$(10) is a common code for activating a line feed. In the American Standard Code for Information Interchange (ASCII) system, you’ll see that these codes are called CR, BS, and LF, respectively (see the ASCII chart, table 2-3). These single codes are all lower than the ASCII decimal code 32, because that is where the numbering of regular letters, numbers, and punctuation marks begins.

There are many other printer functions that require a double code. And almost always, the first of the two codes is CHR\$(27), for which the ASCII abbreviation is ESC, for “escape.” When a sequence of codes starts with this ESC code, it is telling the printer to ignore (escape from) the usual meaning of the code that immediately follows and execute a special command instead.

For all codes that start with CHR\$(27) and have a second code (and sometimes a third) above 32, the ASCII characters themselves can be used, provided they are in quotes, instead of the decimal code inside the parentheses of CHR\$(). The sequence CHR\$(27)“E” for the Epson printers, for example, is just as effective for turning on boldface (“emphasized”) printing as CHR\$(27)+CHR\$(69), the decimal equivalent. Most of these double codes have been selected by the designers of the printers to remind you of their functions (e.g., the Imagewriter’s “f” for setting forward line feed, “r” for setting reverse line feed). With triple codes, both characters following CHR\$(27) can be placed inside a single pair of quotation marks. For example, to switch to proportional letter widths on the Epsoms, CHR\$(27)“pl” can be used instead of CHR\$(27)CHR\$(112)CHR\$(48).

Programmers working with the IBM-Epson and Apple-type printers use these often-mnemonic abbreviations instead of decimals a great deal. This is not only because it’s

easier to associate and remember the codes and functions, but also because it cuts down on the amount of typing. Unlike the printers in these two families, the Tandy printers have very few codes above 32, so their manuals advise you to stick to decimals and, for clarity, separate the CHR\$()'s with semicolons. Even with the few codes above 32, they'll tell you to use CHR\$(27);CHR\$(50) for setting a 1/2-inch line feed, but CHR\$(27)"2" will do just as well. With most combinations of computers and printers, including Tandy com-

TABLE 2-3
ASCII Codes 0-125

Dec	Hex	Character or Function	Dec	Hex	Character or Function	Dec	Hex	Character or Function
0	00	none	42	2A	*	84	54	T
1	01	none	43	2B	+	85	55	U
2	02	none	44	2C	, (comma)	86	56	V
3	03	none	45	2D	- (en dash)	87	57	W
4	04	none	46	2E	. (period)	88	58	X
5	05	none	47	2F	/	89	59	Y
6	06	none	48	30	0	90	5A	Z
7	07	BEL	49	31	1	91	5B	[
8	08	BS	50	32	2	92	5C	\
9	09	HT	51	33	3	93	5D]
10	0A	LF	52	34	4	94	5E	^
11	0B	VT	53	35	5	95	5F	_
12	0C	FF	54	36	6	96	60	` (tick)
13	0D	CR	55	37	7	97	61	a
14	0E	SO	56	38	8	98	62	b
15	0F	SI	57	39	9	99	63	c
16	10	none	58	3A	:	100	64	d
17	11	DC1	59	3B	;	101	65	e
18	12	DC2	60	3C	<	102	66	f
19	13	DC3	61	3D	=	103	67	g
20	14	DC4	62	3E	>	104	68	h
21	15	none	63	3F	?	105	69	i
22	16	none	64	40	@	106	6A	j
23	17	none	65	41	A	107	6B	k
24	18	CAN	66	42	B	108	6C	l
25	19	none	67	43	C	109	6D	m
26	1A	none	68	44	D	110	6E	n
27	1B	ESC	69	45	E	111	6F	o
28	1C	none	70	46	F	112	70	p
29	1D	none	71	47	G	113	71	q
30	1E	none	72	48	H	114	72	r
31	1F	none	73	49	I	115	73	s
32	20	␣ (space)	74	4A	J	116	74	t
33	21	!	75	4B	K	117	75	u
34	22	"	76	4C	L	118	76	v
35	23	#	77	4D	M	119	77	w
36	24	\$	78	4E	N	120	78	x
37	25	%	79	4F	O	121	79	y
38	26	&	80	50	P	122	7A	z
39	27	' (apostrophe)	81	51	Q	123	7B	{
40	28	(82	52	R	124	7C	(vertical bar)
41	29)	83	53	S	125	7D	}

puters driving Tandy printers, it doesn't matter a whit whether you put in the semicolons or leave them out.

It also doesn't matter whether you use plus signs instead of semicolons; `LPRINT CHR$(27)+CHR$(31)` will turn on boldface printing as readily as `LPRINT CHR$(27);CHR$(31)`. We will be following the Tandy convention with semicolons in this book, because the code sequences are easier to read when the semicolons are placed between all those `CHR$()`'s. Plus signs will be used in abbreviating code sequences in the text of the book and for concatenation of strings in BASIC programs.

With the IBM-Epson and Apple printers there are times when you'll want to use decimals instead of ASCII characters in quotes. This is especially the case when the printer codes themselves are values of a numerical variable. If the program contains several `LPRINT CHR$(27);CHR$(A)` or `LPRINT CHR$(27)+CHR$(A)` statements to set different pitches, and the values of `A` are supplied by `DATA` statements, for example, those `A` values would have to be decimals (or be converted from string to numerical values by the `VAL` function) for the pitch-selection command to be executed properly.

Of the various printer functions listed in table 2-2, the most important ones for bit-graphics programming are print density (pitch), boldface printing, line feeds, and horizontal positioning. And there are some important differences between printer families to note in regard to these functions.

Print density. For printers in the IBM-Epson family, *pitch* is a term that usually refers to regular text mode, not to the bit-image graphics mode. So the control codes for changing from standard 10-CPI (the default pitch) to condensed 16.7-CPI printing—the sequence `CHR$(27)CHR$(15)` or `CHR$(15)` alone—change the print density only for printing regular letters, numbers, punctuation marks, and built-in graphics characters outside of the graphics mode. The first table in the next chapter will show that these printers have very different codes for setting or changing print density in the graphics mode; for example, the sequence `27+“L”+N1+N2` for setting a density of 120 dots per inch on the IBM and Epson printers. In other words, the print density (pitch) set in text mode does not automatically carry over to the bit-graphics mode in these two families.

But with Tandy and Apple printers, there is a complete carryover. Whatever pitch has been selected in text mode for regular character printing will also determine the print density in graphic mode. In fact, once you are in graphic mode on the Tandy printers, the only way you can change the print density is to go back to the text mode by the exit-graphic code, `CHR$(30)`, and follow that with your choice of standard (27+19), elite (27+23), condensed (27+20), or elongated (27+14) pitches. This may sound like a disadvantage, but often the carryover from text to graphic mode helps to keep the programming simpler.

Figure 2-5 demonstrates this carryover on the Radio Shack DMP-400 printer, using an inexpensive commercial software package called Dotplot-80. The programs in this package, designed by J. Anthony Cervantes for the TRS-80 Models I, III, and 4, print bit-image renditions of mathematical functions, computer art, and graphs on five different Radio Shack printers.

The upper part of figure 2-5 shows a “nested squares” figure that was produced by Dotplot-80 on the DMP-400 in standard 10-CPI (60-dots-per-inch) pitch. The same program produced the lower figure, except that running the program was preceded by `LPRINT CHR$(30);CHR$(27);CHR$(20)`. This switched the pitch to condensed 16.7 CPI in text mode and also changed the print density in graphic mode to 100 dots per inch. This squeezed in the horizontal printing without affecting the vertical line feeds, resulting in a flattened figure. If you turn the page ninety degrees, you can see that the change in density converted the nested squares to nested diamonds.

Bold printing. There’s another carryover feature in the codes for the Radio Shack printers, and the Apple Imagewriter as well, that is a big advantage in bit graphics—boldface printing. Unlike the IBM and Epson printers, which have emphasized, double-strike, and bold printing in the text mode only, most of the Tandy and Apple printers also provide bold printing in the bit-image graphics mode. If you don’t have a well-inked ribbon in your printer, this makes an enormous difference in the quality of graphic printouts. For a demonstration, see figure 2-6.

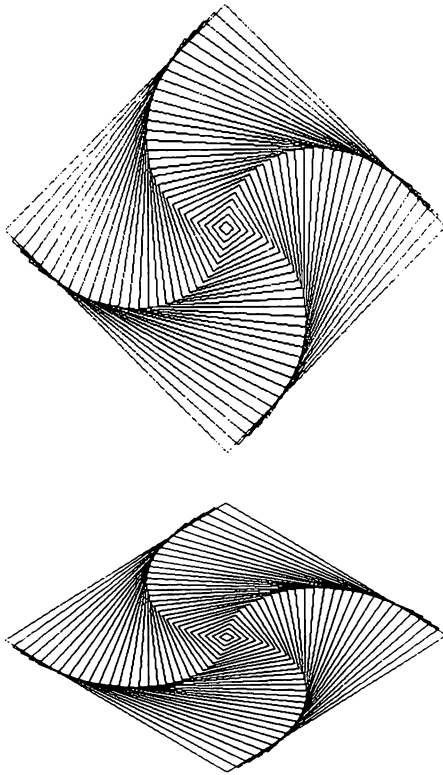


FIGURE 2-5. A demonstration of how printing density, set in text mode, carries over to graphics mode on TRS-80 printers.

Once you get spoiled by the better printing quality with bold-printed graphics, it can be quite a comedown to run similar programs on the IBM or Epson printers. But there are two remedies that will enable users of the latter to pull up close to even with the TRS-80 users. One is to use a new or freshly reinked ribbon (a reinking machine like the Mac Inker, sold by Computer Friends, 100 N.W. 86th Ave., Portland, OR, 97229, is well worth the fifty-five- to seventy-dollar cost).

The other remedy is to use high-density printing in the graphics mode as much as possible. This means going to double density (120 dots/inch) on the IBM or Epson MX-80 printers, and to that same density or even quadruple density (240 dots/inch or 1920 dots/line) on the Epson FX-80 and RX-80 printers. (See the next chapter for the graphic codes controlling these densities.)

Line feeds. About the only size of line feed you'll ever use outside of graphic mode is the $\frac{1}{6}$ -inch (or $\frac{12}{72}$ -inch) feed that

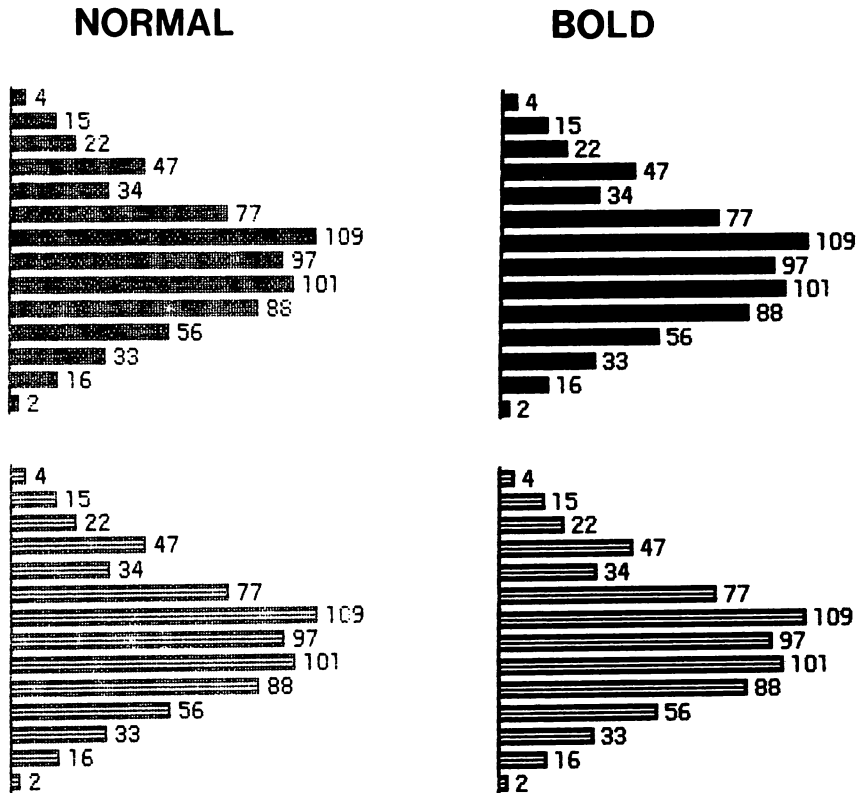


FIGURE 2-6. How printing quality is improved by bold printing in graphic mode with printers having that feature. The same well-worn ribbon was used on the DMP-400 for printing both the left and right figures.

spaces lines of text so that they adhere to the standard six lines per inch of typewriters. This nicely accommodates dot-matrix letters that are usually nine dots high (taking $\frac{9}{16}$ inch) combined with a three-dot-sized ($\frac{3}{16}$ -inch) blank space between lines for good legibility.

An exception to using this standard line feed in text mode is when you are programming a diagram with the built-in line and block graphic characters. In order for these to fit together properly, half-line spacing ($\frac{1}{12}$ inch or $\frac{9}{16}$ inch) is needed when (as is common) the graphic characters are only six dots high, so that there won't be any blank space between the print lines vertically. Other spacing is sometimes needed, such as $1\frac{1}{2}$ - and double-spaced text, but these are

more likely to occur in word processing (and be handled by the word-processing software) than in BASIC programming.

And yet many other line-feed sizes are provided to the programmer by some dot-matrix printers— $\frac{7}{72}$ inch, $\frac{8}{72}$ inch, $\frac{9}{72}$ inch, $\frac{1}{72}$ inch, $\frac{1}{144}$ inch, $\frac{1}{216}$ inch, plus the freedom to set N at whatever you want in $\frac{N}{72}$ -inch, $\frac{N}{144}$ -inch, and $\frac{N}{216}$ -inch sizes. Who needs these? You will, if you do a lot of bit-graphics programming. As we'll see in more detail in many later chapters, you'll need the $\frac{7}{72}$ -inch feed if you're printing dot patterns seven dots high and you want the lines of print to butt together vertically. And you'll want an $\frac{8}{72}$ -inch feed if the patterns are eight dots high, and $\frac{9}{72}$ inch if, as is possible on the FX-80, the patterns are nine dots high. (By now you have probably caught on to the fact that a single dot is $\frac{1}{72}$ inch high, and this too is standard for nearly all printers.)

The tinier line feeds can be helpful in bit-image graphics if you have to make small adjustments of larger feeds for more exact butting of lines of print. More importantly, they are essential if you want to draw in what we call ultra-high-resolution graphics. In this "mode," a drawing is made—slowly—by dozens or even hundreds of passes of the print-head, in which only a single dot is printed and the horizontal dot patterns are "spaced" only $\frac{1}{72}$ inch, $\frac{1}{144}$ inch, or $\frac{1}{216}$ inch "apart" (they would actually be touching or overlapping). Many printers offer enough large and small sizes of line feed that you can program just about any oddball size you want by combining or repeating them.

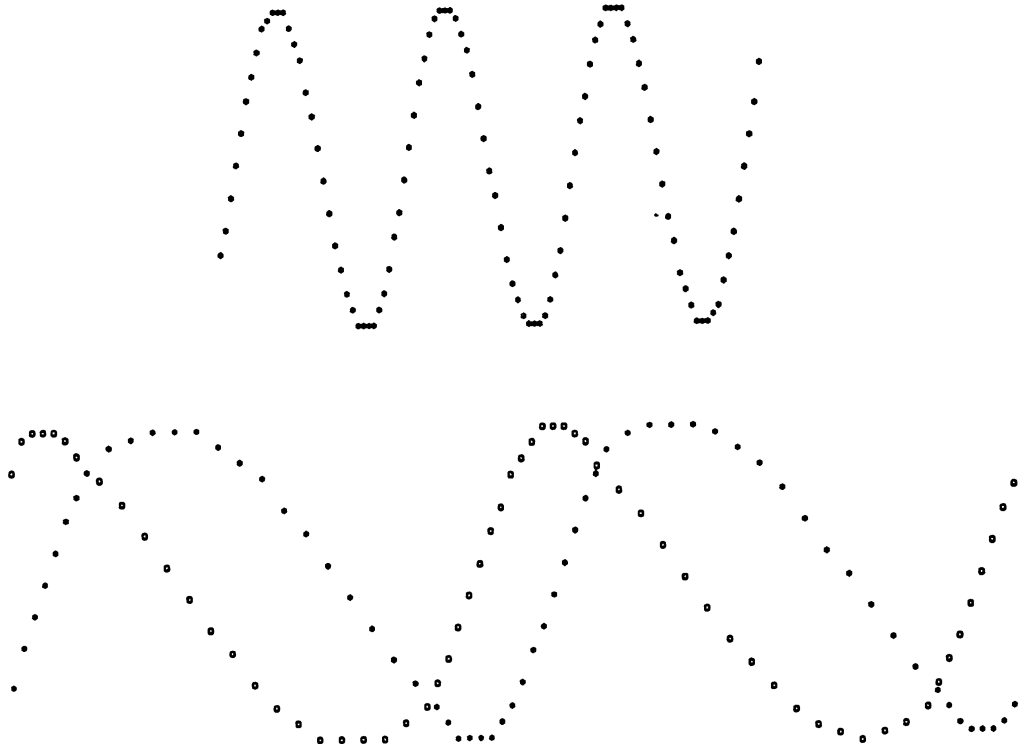
Sometimes reverse line feeds are needed in graphics as well as all these forward feeds. One of the hardest parts of graphics programming—labeling graphs—can be made much easier with reverse feeds. These give you a way of backing up vertically, printing a few characters, and backing up again to return to where you started. But surprisingly, some printers offer only forward line feeds, leaving you with no way to back up vertically.

The most famous of these is the very popular Epson MX-80 (with or without Grafrax). Epson has corrected this deficiency with the newer FX-80 and RX-80, but for some strange reason not the fifteen-inch versions of those, the FX-100 and RX-100. The Transtar 315 Color Graphics and Okidata Micro-line 82 printers also lack the reverse line feed. Many printer-

graphics programmers would not dare to buy these forward-only machines.

Horizontal tabbing. There are also big differences among printers in horizontal tab functions. While nearly all 9.5-inch printers respond to `LPRINT TAB(N)` in much the same way as characters are positioned on the computer's screen by `PRINT TAB(N)`, many printers also have special control codes for presetting a series of tab stops across the page. These are usually activated by `CHR$(9)`. The Radio Shack printers, however, have no horizontal presetting of tab stops. And yet they are even easier to program for horizontal positioning, especially in graphics mode, because they have a direct positioning command, `27+16+N1+N2`, which moves the

FIGURE 2-7. Upper portion: sine curve produced by text-mode positioning using the TAB function (TABMATH1 program). Lower: sine and cosine curves printed simultaneously using the `CHR$(27)"D"...` tabbing function (TABMATH2 program).



printhead precisely to the dot column specified by the combination of N1 and N2.

The Apple Imagewriter, NEC, and C. Itoh printers have an even simpler code for this function, 27 + "F" + N, where N can range from 0000 to 9999. But the IBM and Epson printers do not offer this direct and more exact way of addressing a particular dot column. We'll see in the next chapter that this often makes positioning with these printers rather inconvenient, even for computers (such as the IBM PC) which have the BASIC function SPACE\$, another method of horizontal tabbing in text mode.

This is as good as any place to demonstrate some kinds of printer graphics that can be done entirely outside of graphics mode using horizontal tabbing. Even though these are confined to low resolution, it's good to know that they're available for some purposes. In program 2-1 (TABMATH1), the TAB() feature is used by an IBM PC to draw a sine wave on the Epson FX-80 (the top of figure 2-7). This is simply a matter of putting a variable, N, in the parentheses of TAB(A) and letting A's value be determined by the SIN() function. An unusually small line feed for text mode, $\frac{5}{72}$ inch, was used to compress the drawing into steeper curves.

PROGRAM 2-1. TABMATH1. Plotting a sine curve using the TAB function in text mode.

```

10 'PROGRAM 2-1: "TABMATH1" -- SINE CURVE DRAWING (IBM PC AND EPSON FX-80)
20 OPEN "LPT1:" AS #1: WIDTH #1, 255      'SET PRINTER WIDTH TO 'INFINITY'
30 PRINT #1, CHR$(27)"@";                  'MASTER RESET
40 PRINT #1, CHR$(27)"A"CHR$(5)           'SET 5/72" LINE FEED
50 FOR N=-10 TO 10 STEP .2
60   A=40-INT(20*SIN(N)): PRINT N, A      'SINE FUNCTION
70   PRINT #1, TAB(A)**                    'PRINT ASTERISK AT TAB STOP A
80 NEXT N
90 CLOSE #1: END

```

The second example of horizontal tabbing (program 2-2, TABMATH2) uses the IBM PC and IBM Graphics printer so that two different text-mode graphic characters can serve as data points for two curves at a time (see the bottom of figure 2-7). Instead of TAB(A), the horizontal tab-setting code, CHR\$(27)"D"CHR\$(A)CHR\$(B)CHR\$(0), and the tab-activating code, CHR\$(9), are used to position the data points.

PROGRAM 2-2. TABMATH2. Plotting sine and cosine curves using the 27+“D” horizontal positioning command.

```

10 'PROGRAM 2-2: "TABMATH2" -- SINE AND COSINE CURVES (IBM PC & EPSON FX-80)
20 OPEN "LPT1:" AS #1: WIDTH #1, 255      'SET PRINTER TO 'INFINITE" WIDTH
30 PRINT #1, CHR$(27)"a";                  'MASTER RESET
40 PRINT #1, CHR$(27)"A"CHR$(3);           'SET 3/72" LINE FEED
100 FOR N=-10 TO 10 STEP .2
110   A=40-INT(20*SIN(N))                  'SINE FUNCTION
120   B=40-INT(20*COS(N))                  'COSINE FUNCTION
130   PRINT N,A,B                          'DISPLAY VALUES ON MONITOR
140   IF B>A THEN PRINT #1, CHR$(27)"D"CHR$(A)CHR$(B)CHR$(0);: PRINT #1, CHR$(9)
      "*"CHR$(9)"o"                        'PLOT A (*) AND B (o) VALUES
150   IF A>=B THEN PRINT #1, CHR$(27)"D"CHR$(B)CHR$(A)CHR$(0);: PRINT #1, CHR$(9)
      )"o"CHR$(9)"*"                      'PLOT B FIRST IF A EQUALS OR EXCEED B
160 CLOSE #1: END

```

A and B are two different tab stops determined by sine and cosine functions, respectively. Since A is greater than or equal to B some of the time and B is greater than A the rest of the time, two different CHR\$(27)“D”...CHR\$(9) statements are needed (lines 140 and 150 in program 2-2). These could be replaced by TAB(A) and TAB(B) or SPACE\$(A) and SPACE\$(B) statements for simplicity’s sake, although with some systems it is risky to have two (or more) TABs on the same print line when the order of TAB(A) and TAB(B) is continually changing.

In these examples PRINT#1 (preceded by an OPEN “LPT1:” AS #1 statement) was used instead of the simpler LPRINT to guard against “trouble codes” delivered to the printer by the IBM PC’s software. When LPRINT is used, for example, an A or B value of 13 triggers an unwanted line feed in the midst of printing a line. A good deal of space in the next chapter is devoted to ways of detecting and dealing with such trouble codes, since they pose many more problems in the bit-graphics mode than in the text mode.

Finer Points

In case you’re thinking that we’ve covered most of the features found in dot-matrix printers, look in your printer manual. There are another two dozen or so functions, many with special control codes, which we haven’t mentioned yet. And again we find large differences among printers in their offerings. In most cases, these finer features will be explained as

needs arise in later chapters. But some miscellaneous comments about a few of them are needed at this point.

Printing speed. If you're only acquainted with typewriters, dot-matrix printers will dazzle you with their speed. Despite the fact that up to ten firings of the printhead pins may be required to print just one character in text mode, many printers can print over one hundred characters per second. In bit-image mode, printing over five hundred dot patterns per second is commonplace.

Often it hardly matters how fast a printer can print in bit-image graphics, because the printer is spending a lot of time waiting for the computer to deliver the codes. This is especially likely to happen when the programming contains large arrays, many FOR-NEXT loops, poorly placed subroutines, and so many strings that the computer's string storage space in RAM must be reorganized in the midst of a program run.

DIP switches. Every printer has a set of manual function switches for presetting or changing such things as international alphabets, baud rates in serial interfacing, and control of line feeds and carriage returns. These are called DIP (dual in-line package) switches. By far the most important of these for bit-image graphics is the line feed (LF) and carriage return (CR) control. Unlike text printing, bit-graphics printing often requires several different commands for the same line of print, including CR's, which may be located at any point, and sometimes at more than one point within a series of BASIC statements. Many computers are designed to deliver automatic LF commands with every CR or end-of-line signal in text mode, and carry this over to graphic mode, producing a lot of unwanted line feeds. Often the best way to remedy this is to set the CR-LF DIP switch to the position that will prevent all line feeds until a specific LF code, such as CHR\$(10), is received by the printer. As we'll see, things can get even more complicated when the computer automatically converts all CHR\$(10) codes to the CR code, CHR\$(13), as with some TRS-80 computers.

Buffer space and operation. Dot-matrix printers do not immediately print each character code (ASCII or bit-image) as it is received. They all have a temporary storage space, called

the print buffer, in which many codes are collected before any one of them is printed. Some print buffers can hold over two thousand bytes of such information (including control codes as well as character or dot-pattern codes). While holding the information in the buffer, the printer also organizes it into lines of print. Usually the signal for the buffer's contents to be printed comes at the occurrence of a CR code or at the end of a print line, when a CR code is automatically generated. Emptying the buffer (by printing its contents) can also occur when the buffer is full, or when certain other control codes are received; the manual for your printer should indicate what events trigger the printing from the buffer. In misbehaving bit-graphics programs, knowing these conditions can sometimes be crucial to debugging, especially in high-density printing on fifteen-inch printers.

Custom characters. A few of the more recent printers, including the FX-80 and Apple Imagewriter, allow you to design your own dot-matrix characters—letters, numbers, and punctuation marks—and store them in the printer's memory. This can be very useful in graphics programming, again because of challenges that arise in the labeling of graphs or diagrams. These user-defined characters could even be upside down, which might be just what you need for labeling the vertical axis of a graph. They can also be just parts of letters, which can be combined with other parts to form extra-large letters. Chapter 7 goes into this topic deeply, explaining how to make custom alphabets even if your printer doesn't have special codes or storage space for doing this as the FX-80 and Imagewriter do. (The homemade characters can be stored in the computer instead of the printer.)

The lesson, then, is: Learn everything you can about your printer's nongraphic features, and you'll be in a position to accomplish much more in bit-image graphics.

Welcome to Dotland!

IF you've never been there before, get ready for a few jolts when you enter the world of bit-image graphics. In text mode, you're used to seeing the printhead move and hearing it impact with every character printed. The first time you try to make it print a dot or dot pattern you may not see or hear anything—even if you succeed.

Here is the likely scenario. You send a print command to the printer and nothing happens. You try a second time—nothing again. It quickly becomes a situation like telekinesis—maybe if you *will* it hard enough, you think, you can get that printhead to budge! You might well have printed nothing in those first two attempts—there are lots of ways to get neither a text character nor a dot pattern after typing, say, `LPRINT CHR$(72)`.

But look again: You probably printed a dot pattern but can't see it because the printhead is in the way. And if you'd been attending closely to the printer when you hit the ENTER key, you probably would have heard a small sound and seen the printhead move a tiny distance. Trouble is, printing a single dot pattern in normal density takes only one-sixth the horizontal distance that printing a letter or number does. So you've entered a world of graphics that operates on a miniature scale.

But let's suppose that after cranking up the paper by hand for a better look, you find that there is not a single black dot on it. It could be that you didn't use the right control codes to get into graphic mode in the first place. More likely, per-



FIGURE 3-1. Two-inch bar, produced on the DMP-400 by the command `LPRINT STRING$(120,255)`.

haps you entered graphic mode all right, but didn't send as many dot codes to the printer as it was expecting—many printers have to be told how many dot columns of space in a line to reserve for bit-image codes as part of the coding for entering graphic mode. If you don't match the number of dot-columns with the number of dot codes you send, the printer usually won't be very cooperative.

The Epson printers can be especially punishing to the user who sends too few or too many dot codes. They will cough, sputter, hiccup, sound their beepers (a pretty good imitation of R2D2, in fact), disobey the next commands you send, or sit in ominous silence, seemingly according to their mood. This business of having to match the amount of graphic space reserved is, as we'll see, a major difference between printers that can also be a major headache at times.

A better way to start in bit-image graphics is to try some repeated codes, so that most of what you print won't be hidden behind the printhead. With an IBM system, `LPRINT CHR$(27)"K"CHR$(120) CHR$(0) STRING$(120,255)` will give you a solid bar that is two inches long and eight dots high, as in figure 3-1.

With an Apple system using Applesoft BASIC the equivalent command would be:

```
PR#1: PRINT CHR$(27);CHR$(83);
      "0144";CHR$(27);CHR$(86);"0144";CHR$(255)
```

or

```
PR#1: PRINT CHR$(27)"S0144": FOR N=1 TO 120:
      PRINT CHR$(255): NEXT N
```

(but it may be necessary to use a POKE command in place of PRINT to get the two-inch bar—see the section on trouble codes later in this chapter). Tandy users should try `LPRINT CHR$(18);STRING$(120,255)`.

Better yet, get a look at the whole range of dot patterns your printer can print with the statements in programs 3-1A, 3-1B, and 3-1C.

These three sets of commands will reveal the dot patterns

PROGRAM 3-1A. DOTPRINT/EPS. Printing the entire range of dot codes for IBM-Epson printers.

```
10 'PROGRAM 3-1A: "DOTPRINT/EPS" -- FOR IBM PC AND IBM-EPSON PRINTERS
20 OPEN "LPT1:" AS #1: WIDTH #1, 255      'SET PRINTER TO 'INFINITE' WIDTH
30 PRINT #1, CHR$(27)"K"CHR$(255)CHR$(0);  'RESERVE 255 COLS. FOR GRAPHICS
40 FOR I=1 TO 255
50   PRINT #1, CHR$(I);                    'PRINT DOT-PATTERN FOR DOT-CODE I
60 NEXT I
```

PROGRAM 3-1B. DOTPRINT/APL. Printing the entire range of dot codes for Apple printers.

```
10 REM -- PROGRAM 3-1B: "DOTPRINT/APL" -- FOR APPLE IIe AND IMAGEWRITER
20 PR#1
25 PRINT CHR$(27)+"p";: REM -- 160 DOTS PER INCH DENSITY
30 PRINT CHR$(27)+"G0256";: REM -- RESERVATION OF 256 DOT COLUMNS
40 FOR I=0 TO 255
50   IF PEEK(49305) <> 16 THEN GOTO 50
55   POKE 49304,I: REM -- SEND DOT CODE TO PRINTER
60 NEXT I
```

PROGRAM 3-1C. DOTPRINT/TRS. Printing the entire range of dot codes for Tandy printers.

```
10 'PROGRAM 3-1C: "DOTPRINT/TRS" -- FOR TRS-80 COMPUTERS AND PRINTERS
20 LPRINT CHR$(18);      'ENTER GRAPHIC MODE
30 FOR I=129 TO 255
40   LPRINT CHR$(I);     'PRINT DOT-PATTERN FOR DOT-CODE I
50 NEXT I
```

in a compressed fashion, as in figure 3-2. In the Apple example (program 3-1B), the use of PRINT for sending control codes and POKE (with PEEK) for sending dot codes is demonstrated.

Notice that the IBM-Epson and Apple Imagewriter samples are eight dots high and show twice as many patterns as the Tandy printout, which is seven dots high. Notice also that all 255 of the IBM-Epson dot patterns are “upside down” (mirror images) with respect to the Imagewriter dot patterns, and the first half of the IBM-Epson patterns are mirror-images of the Tandy DMP-400 patterns. The next chapter provides a close-up of each dot pattern on each printer.

You can spread out the dot patterns by putting a blank code—CHR\$(0) on the IBM-Epson and Imagewriter printers, CHR\$(128) on the Tandys—after CHR\$(I) in the print commands above. But before running on the IBM and Imagewriter, you’ll have to double the number of dot columns to be reserved for bit graphics. This means changing CHR\$(0) to CHR\$(1) in line 30 of the IBM program and changing

IBM-EPSON**APPLE****TRS-80**

FIGURE 3-2. Printouts of all dot codes by Epson, Apple, and Tandy printers.

“G0256” to “G0512” in line 30 of the Apple program. The Tandy printers don’t care how many columns of space you need—once you put them in graphic mode they’ll just stay there until you send `CHR$(30)` to exit and return to the text mode.

Before trying to do much more, you’d be wise to learn all the control codes for bit-image graphics.

Graphic-Mode Control Codes

Table 3-1 is a summary of all the control codes available in graphic mode on the Epson FX-80, the Apple Imagewriter, and the Radio Shack DMP-series printers other than the DMP-2100. In the lower part of the table, there are also a few control codes you’re likely to need in graphics but that are not available unless you temporarily exit the graphic mode.

Entering and exiting graphic mode. With the IBM and Epson printers the seemingly simple matter of entering graphic mode turns out to be three different functions controlled by a single four-byte command. The command starts with the escape code, `CHR$(27)`, and is immediately followed by a code that determines which printing density will

TABLE 3-1
Control Codes in Graphic Mode for the Three Printer Families

	IBM-EPSON	APPLE	TANDY
PIN CODING	EPSON FX-80 ● 128 ● 64 ● 32 ● 16 ● 8 ● 4 ● 2 ● 1 (For 9-pin graphics) ● 128 (2nd byte)	● 1 ● 2 ● 4 ● 8 ● 16 ● 32 ● 64 ● 128	● 1 ● 2 ● 4 ● 8 ● 16 ● 32 ● 64
ENTER GRAPHIC MODE	CHR\$(27)*K*CHR\$(N1)CHR\$(N2) CHR\$(27)*L*CHR\$(N1)CHR\$(N2) (*K*=single, *L*= double density)	CHR\$(27)*Gnnnn* or CHR\$(27)*Snnnn*, where nnnn=8888-9999	CHR\$(18)
EXIT GRAPHIC MODE	(Automatic after reserved graphic dot-columns are filled)	(Automatic after reserved graphic dot-columns are filled)	CHR\$(38)
PRINT DENSITY (dpi=dots/inch)	CHR\$(27)*D*CHR\$(N1)CHR\$(N2), where D=8 (60 dpi), 1 (120 dpi, low speed), 2 (120 dpi, high speed), 3 (240 dpi), 4 (80 dpi), 5 (72 dpi), or 6 (90 dpi). 9-pin graphics: CHR\$(27)*D*CHR\$(N1) CHR\$(N1)CHR\$(N2), where D=8 (60 dpi) or 1 (128 dpi)	72 dpi (576 dpi): CHR\$(27)*n* 80 dpi (640 dpi): CHR\$(27)*N* 96 dpi (768 dpi): CHR\$(27)*E* 120 dpi (960 dpi): CHR\$(27)*q* 136 dpi (1,088 dpi): CHR\$(27)*Q* 144 dpi (1,152 dpi): CHR\$(27)*p* 160 dpi (1,280 dpi): CHR\$(27)*P*	CHR\$(27);CHR\$(19): 60 dpi CHR\$(27);CHR\$(23): 70 dpi CHR\$(27);CHR\$(28): 100 dpi (Must be set in text mode)
DIRECT POSITIONING	NONE (use repeat-blanks)	CHR\$(27)*Fnnnn*, where nnnn = 8888- 1,260 dot-columns from left margin	CHR\$(27);CHR\$(16);CHR\$(N1);CHR\$(N2) (N1=8-5, N2=8-255)
REPEAT PRINT	NONE (use STRING\$ or FOR-NEXT loop)	CHR\$(27)*Vnnnn*, where nnnn = 8888- 1,268 (or STRING\$(N,DOTCODE))	CHR\$(28);CHR\$(N) or STRING\$(N,DOTCODE) (N=8-255)
BACKSPACE	CHR\$(8), in text mode	CHR\$(8)	CHR\$(8);CHR\$(N) (In text mode)
PROPORTIONAL SPACE	NONE	CHR\$(27)CHR\$(N) (N=1-6)	CHR\$(27);CHR\$(N) (In text mode, N=8-9)
7/72" LINE FEED	CHR\$(27)*1", activated by CHR\$(18)	CHR\$(27)*T14"	(Automatic in graphics mode)
N/72" LINE FEED	CHR\$(27)*A*CHR\$(N), act. by CHR\$(18)	NONE	CHR\$(27);CHR\$(58) = 1/72" (Repeat for higher N values)
N/144" LINE FEED	NONE	CHR\$(27)*Tnn", where nn=88-99	NONE
N/216" LINE FEED	CHR\$(27)*3*CHR\$(N), act. by CHR\$(18) or CHR\$(27)*J*CHR\$(N), executive	NONE	CHR\$(27);CHR\$(51) = 1/216" (Repeat for higher N values)

be in effect. After that comes a pair of codes controlling the number of dot columns being reserved for bit-image graphics. With the IBM Graphics and MX-80 printers, the choice of densities is limited to "single" (60 dots/inch, 480 dots/line) and "double" (120 dots/inch and 960 dots/line). For single density the second of the four bytes must be "K" or CHR\$(75) and for double density the code is "L" or CHR\$(76).

The pair of codes determining the dot-column space must be in a particular order and neither may exceed 255, because the CHR\$() function in BASIC is limited to the range of 0 to 255. If the number of dot columns to be reserved is under

256, then the first code in the pair must equal the number reserved, and the second code must be zero. Beyond 255, the second code must be greater than zero, and you have to use a combination of the two codes to reserve the number of dot columns you want.

The second of the two codes must represent the number of whole 256-dot units the reservation number contains. So second codes of 1, 2, and 3 stand for dot-column lengths of at least 256, 512, and 768, respectively. Then it's up to the first code in the pair to make up the difference: if the number to be reserved is 300, then the first code must be CHR\$(44) and the second CHR\$(1)—44 plus 256 equals 300. If the number is 960, then the pair must be CHR\$(192)CHR\$(3), 192 being the difference between 960 and 768, and 3 being the number of whole 256-dot units contained in 960.

The basic form of the "graphics-density-space" command, then, is CHR\$(27)"K"CHR\$(N1)CHR\$(N2) or CHR\$(27)"L"CHR\$(N1)CHR\$(N2) for single and double densities, respectively, with the sum of N1 and 256-times-N2 determining the total number (T) of dot columns reserved. You can get the values of N1 and N2 for any value of T by a series of IF-THEN statements, as follows:

```
200 IF N<256 THEN N2=0: N1=N
210 IF N>255 AND N<512 THEN N2=1:
    N1=N-256*N2
220 IF N>511 AND N<768 THEN N2=2:
    N1=N-256*N2
```

and so on. But it's much shorter this way:

```
200 N2=FIX(N/256): N1=FIX(N-256*N2)
```

In general things will work better if N1 and N2 are in integer form, which is what FIX guarantees. FIX is safer to use than INT or CINT because with some computers (e.g., the TRS-80 Model 4) single- and double-precision numbers are rounded rather than truncated by these latter functions.

The Epson FX-80 has five more densities than the MX-80 or IBM printers. With so many choices, the basic command becomes CHR\$(27)"*"CHR\$(D)CHR\$(N1)CHR\$(N2), with "*" used for all densities and D selecting densities that are numbered from 0 to 6. The value of D for single density is 0,

and for ordinary double density it is 1. Then the FX-80 goes on to a “high-speed” double density (D equals 2), quadruple density (240 dots/inch, 1,920 dots/line!) when D is 3, and three more densities that are between single- and double-density dot packing.

One of these latter deserves special attention. When D equals 5, the density is 576 dots/line, or 72 dots/inch (there’s that number 72 again). This is the setting that gives you a one-to-one ratio of horizontal-to-vertical dot density; recall that a single dot is $\frac{1}{72}$ inch. This provides the most direct way of bit-mapping a graphic creation that will print out with no horizontal-vertical distortion, which is often so obvious a problem in both screen and printer graphics. So three cheers to Epson for this one-to-one (or if you wish, 72-to-72) feature on the FX-80. (The RX-80 has the same set of densities as the FX-80, except for this one-to-one density.)

After all this work getting into graphic mode with an IBM-Epson printer, what do you have to do to get out? Nothing, it turns out. As soon as the printer has received the number of dot codes that it takes to fill all the dot columns that were reserved, it automatically shifts back to text mode. If you want to print some more graphics, you have to make another reservation of dot-column space. This works the same way on the MX-80, RX-80, and IBM Graphics printers.

The Imagewriter and other printers in the Apple family (C. Itoh 8510, NEC 8023A, ADS 8001) are like the IBM-Epson printers in requiring a space reservation as part of the command for entering graphic mode, but this is not done with a pair of numbers. Instead, a single four-digit code ranging from 0000 to 9999, enclosed within quotation marks, will reserve the number of dot columns needed at a time. And unlike the rules for the IBM and Epson printers, this number is not restricted by the total number of dot columns in a single line (which is 1,280 at the highest bit-graphics density available on the Imagewriter, 160 dots per inch). So if you want to print six consecutive lines of graphics, you can specify “7680” (1,280 times 6) to reserve the dot-column space in a single statement. It’s usually easier to reserve graphic space on a line-by-line basis, as with the IBM-Epson printers.

The “enter graphic mode” command for the Apple-family printers is relatively simple. With the Imagewriter, the key codes are either “G” or “S” (which are completely inter-

changeable—either starts the graphic reservation). For example, after `PR#1` the statement `PRINT CHR$(27); "G"; "0560"` would amount to telling the printer, "I want to send dot codes requiring 560 dot columns of graphic space." `PRINT CHR$(27); "S"; "0560"` would amount to the same. You could make these even shorter by dropping the semicolons and combining the two quotes into one: `PRINT CHR$(27)"G0560"` or `PRINT CHR$(27)"S0560"`. Such efficiency should be appreciated, because so many other things in graphics require a lot of typing.

Incidentally, neither the IBM-Epson- nor Apple-type printers require you to put everything in one line of your BASIC program. As long as the printer receives the codes in the correct order, it doesn't care whether you do it all at once or spread the "enter graphics," "density," and "reserve space" parts over three different LPRINT lines in the program. With Apple printers, in fact, the density must be specified (see table 3-1) in a separate statement that precedes the graphic reservation.

Entering graphics mode on the Radio Shack printers is even simpler—just type `LPRINT CHR$(18)`. But that says nothing about print density. To select among the pica, elite, condensed, (and their combinations with elongated) that are offered, you have to send the appropriate pair of codes—for example, `LPRINT CHR$(27); CHR$(20)`; for condensed—before you enter graphic mode. If you're already in graphic mode, exit by inserting

```
CHR$(30);
```

just after LPRINT and reenter by putting

```
CHR$(18);
```

at the end of the density command (or in a new LPRINT line). Again, `CHR$(30)` is the code for getting out of graphic mode, and it's good to know this by heart, because the printer will absolutely refuse to print regular text characters when it's in graphic mode. (You'll usually get a blank line feed, or many of them, instead of a character print.)

Repeating a dot pattern. Any command, in graphics or text mode, can be repeated by enclosing it within a FOR-NEXT loop. But the Apple and Tandy printers provide a more con-

venient and faster way of repeating dot-codes in graphic mode. If we let DC stand for the dot code, all it takes on the Imagewriter is `LPRINT CHR$(27)"Vnnnn"CHR$(DC)` where, again, nnnn is a four-digit number, which can range from 0000 to 9999.

The TRS-80s do their repeating by the code sequence `CHR$(28);CHR$(N);CHR$(DC)`, where N can vary from 0 to 255. As with the Imagewriter's `27+"V"+nnnn+DC` sequence, the `28+N+DC` command repeats the printing of a dot code starting wherever the printhead happens to be at the time the repeat code is received, not necessarily at the left margin. Actually, the `28+N+DC` sequence doesn't get much use when a computer uses Microsoft or other strong BASICs, because it's even easier to use the `STRING$` function for repeating a code. `LPRINT STRING$(N,DC)` does exactly the same as `LPRINT CHR$(28);CHR$(N);CHR$(DC)`, and in both cases N cannot exceed 255. If you want to repeat a code more than that, say, 355 times, you can string the `STRING$` together and use `LPRINT STRING$(255,DC);STRING$(100,DC)`. (You can also string the `28+N+DC` sequences together, as in `LPRINT CHR$(28);CHR$(255);CHR$(DC);CHR$(28);CHR$(100);CHR$(DC)`, but that is twenty-three extra characters to type.) Compared to both the `FOR-NEXT` way of repeating and the `28+N+DC` sequence, `STRING$` speeds things up in the printer's execution of the command as well as the time it takes to type it.

It's a good thing the Apple-family printers have their `27+V+nnnn+DC` command for repeating, because Apple-soft BASIC, used in the Apple II and III computers, doesn't have the `STRING$` function. And when you have an IBM or Epson printer being driven by one of these Apple computers, you're limited to `FOR-NEXT` loops for repeating, because (amazingly) the IBM and Epson printers don't have a control code for repeating a character or dot code. Putting this another way: It's a good thing that IBM and Epson printers are usually driven by computers that do have the `STRING$` function (the IBM PC, most of the TRS-80s, Apple Microsoft BASIC, and the Epson HX-20 have it; the Commodore 64 and VIC-20, and Atari 400 and 800 don't).

Another advantage of `STRING$` with the TRS-80 computers and printers is that there is no trouble when N is 0, as in `STRING$(0,136)`, whereas the printers go haywire when N

is 0 in the $28 + N + DC$ sequence. “Haywire” is a bit exaggerated; what happens is that you get 255 repetitions instead of none (which is bad enough!). The Imagewriter’s repeat command doesn’t have this problem with zeros.

Other control codes. The remaining codes listed in table 3-1 have to do with positioning the printhead and line feeds. The positioning codes, which include the direct-positioning, backspace, forward spacing, and other commands, will be treated in the next section because of the special importance of controlling the printhead’s position in graphics. As for line feeds, there are several points to note beyond what was mentioned in the last chapter.

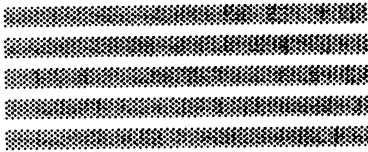
The IBM and Epson printers have all the forward line feeds you should never need, and the Epson FX-80 and RX-80 have a reverse line set of $N/216$ inch, with the value of N up to the user in the range of 1 to 216. The Epson FX-80’s reverse-feed code is an activating rather presetting one—the sequence `LPRINT CHR$(27)"j"CHR$(N)` fires an immediate one-time reverse feed without a carriage return. The TRS-80s also have a reverse feed (in text mode and inconveniently restricted to $1/6$ inch and $1/12$ inch, which can be activating or presetting, depending on other codes preceding the line-feed codes. The Apple Imagewriter and Dot Matrix printers have a reverse-feed set command, `CHR$(27)"r"`.

With some dot-addressable printers, the forward line feed in graphic mode is automatically set to a size that will make successive lines of print butt together vertically to form a continuous image. The Radio Shack printers, dating from the TRS-80 LPVIII printer, have this feature. The height of any graphic line with these printers is normally fixed at seven dots, so the line feed is automatically set to $7/72$ inch. This is the distance the paper will move when the printer receives the universal line-feed activating code, `CHR$(10)`. The butting of graphic print lines is illustrated in figure 3-3.

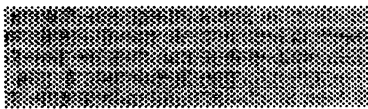
Very small increases in this distance can be made on most of the TRS-80 DMP-series printers by adding $1/216$ -inch line feeds—`CHR$(27);CHR$(51)`—to the `CHR$(10)` code if there is any overlap of the print lines. This can be done without leaving graphic mode, as can the addition of $1/72$ -inch line feeds with the sequence `CHR$(27);CHR$(50)`. Either of these small feeds can be repeated as many times as you want

IBM-EPSON PRINTERS

LINE FEED: $1/6$ INCH



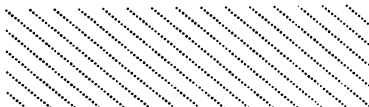
LINE FEED: $8/72$ INCH



LINE FEED: $1/6$ INCH



LINE FEED: $8/72$ INCH



TRS-80 PRINTERS

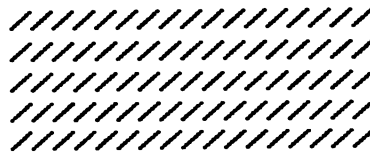
LINE FEED: $1/6$ INCH



LINE FEED: $7/72$ INCH



LINE FEED: $1/6$ INCH



LINE FEED: $7/72$ INCH



FIGURE 3-3. Adjusting the line feed so that adjacent print lines will butt and form a continuous image.

(with FOR-NEXT loops), so oddball line feeds such as $\frac{2}{216}$ inch, $\frac{5}{216}$ inch, and $\frac{11}{72}$ inch can be used. In using these, note that 216 is three times 72, and thus $\frac{3}{216}$ -inch feeds are nominally equal to $\frac{1}{72}$ -inch feeds.

This needs to be checked, though, because the actual size of line feeds, particularly these small ones, may not be precisely consistent when repeated. Don't be surprised if repeating a $\frac{3}{216}$ -inch feed fifty times comes out a quarter-inch longer or shorter in paper movement than repeating a $\frac{1}{72}$ -

inch feed fifty times. Worse yet, you can often tell at a glance from printouts of thin lines separated by supposedly constant line feeds that the spaces between lines are definitely not constant. This seems to be a problem that all of the popular printers have, not just the TRS-80s, and it's the most likely way a printout can be spoiled. Dot-matrix printers have made impressive advances in technology since the 1970s, but this is one area where a lot more work needs to be done.

The IBM and Epson printers also have $\frac{1}{216}$ -inch and $\frac{1}{72}$ -inch line feeds—CHR\$(27)“3”CHR\$(N) and CHR\$(27)“A”CHR\$(N), respectively, when N = 1 in the more convenient $\frac{1}{216}$ -inch and $\frac{1}{72}$ -inch presetting codes. The Epson FX-80 also has an activating, one-time forward feed of $\frac{1}{216}$ inch, for which the code sequence is CHR\$(27)“J”CHR\$(N). In addition, there are presetting codes for $\frac{1}{8}$ inch [CHR\$(27)“0”], $\frac{7}{72}$ inch [CHR\$(27)“1”], and $\frac{1}{6}$ inch [CHR\$(27)“2”]. The $\frac{7}{72}$ -inch feed, for seven-pin graphics, is not automatic with entry into the graphics mode, nor is the setting for eight-pin graphics, which require an $\frac{8}{72}$ -inch sequence—CHR\$(27)“A”CHR\$(8). Line feeds for all preset sizes are activated by CHR\$(10) as usual.

The Apple Dot Matrix and Imagewriter printers are much more limited in the number of different line feeds they offer, but that doesn't hurt very much because each of them offers a flexible $\frac{1}{144}$ -inch feed. The code sequence for this is CHR\$(27)“T”CHR\$(nn) where nn is a two-digit number in the range of 01 to 99. For seven-bit graphics, nn would be 14 (but try 13 or 15 if print lines don't butt well), and for eight-pin graphics nn would be 16 (or 15 or 17). The 27 + T + nn feed can be forward (the default condition), if preceded by CHR\$(27)“f”, or reverse, if that has been preset by CHR\$(27)“r”. Again, CHR\$(10) is the executing command code, for either direction, and for two other line feeds offered, $\frac{1}{6}$ inch and $\frac{1}{8}$ inch. As with the IBM and Epson printers and some of the TRS-80 codes, the printer must be out of graphic mode in order to respond to these line-feed setting or activating codes.

Precise Printhead Positioning

We've now reached what is in many ways the most important topic of this book—how to control the printhead's horizontal

position. This is often not a matter of using a single command, but rather putting together a combination of commands. The ingredients to be combined include the horizontal tab functions, direct positioning (if available), repeat codes, backspacing, and forward spacing, and all of these will usually behave differently, or have to be treated differently, as the print density changes.

Horizontal tab functions. A chapter ago, we saw the TAB(N) and 27+D+N1+N2+...+0 horizontal tabbing features drawing sine waves on an Epson printer. If you're thinking that we'll use these a great deal in bit-image graphics, guess again. These are really low-resolution tools designed for text mode. That is why they operate in jumps of six dot columns at a time—ordinary text characters are six dots wide. If we only wanted to move the printhead to dot-column addresses of 6, 12, 18, and other multiples of six, we would find these tab functions very useful. But in graphics we are much more likely to need addresses between multiples of six, e.g., column 40, between the sixth and seventh tab stops (i.e., the 36th and 42nd dot columns).

Besides this problem, some printers, especially Tandy's DMP-series, will position correctly at tab stops only when the print density is normal; they may do some rather crazy things under elite and condensed densities. Furthermore, with a fifteen-inch printer such as the DMP-400, the TAB(N) function will not perform reliably above the 480th dot column even in standard 10-CPI density. This only helps to prove that the TAB() function was mainly designed for tabbing on the computer's screen rather than on the printer anyway.

The horizontal tabbing functions are not completely useless in bit graphics. They can be combined with other commands to give you the most precise positioning, as we'll see shortly. In fact, with printers that don't offer a direct-positioning command (including the IBM and Epson printers) we're sometimes forced to do a lot of tabbing mixed with other ways of moving the printhead. Also, for such things as numbering an axis on a bar or line graph, where we may not even have to be in graphic mode, both the TAB(N) and the 27+D... horizontal tabbing features can be handy at times.

The direct-positioning command. Radio Shack printers have a way of shifting the printhead's position from any location in a line to any other location. This, in fact, is part of the definition of what we're calling the direct-positioning command. It is by far the most-used graphic feature on those machines.

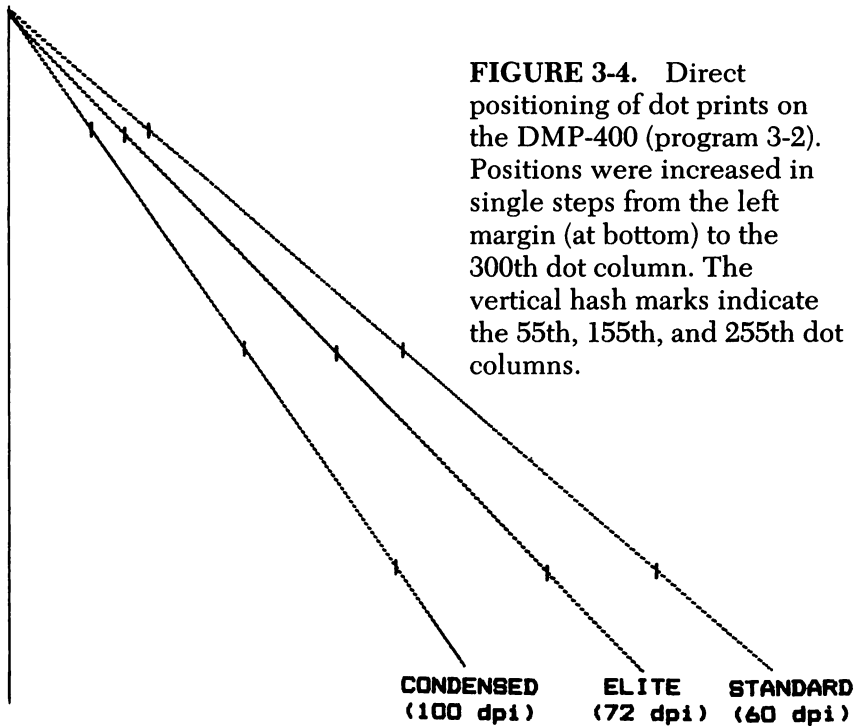
The code sequence for this command is `CHR$(27);CHR$(16);CHR$(N1);CHR$(N2);` (which we'll call the $27 + 16 + N1 + N2$ sequence for short). With this coding, the combination of $N1$ and $N2$ determine the dot-column address to which the printhead moves. As usual, neither $N1$ nor $N2$ can exceed 255, and so we have to work with multiples of 256 and remainders as we did with reserving numbers of dot columns for the graphic mode on IBM-Epson printers.

One big difference, though: $N1$ and $N2$ in the TRS-80's direct-positioning sequence are reversed from $N1$ and $N2$ in the Epson space-reservation sequence, $27 + K + N1 + N2$. The $N1$ in the positioning command is, like $N2$ in the Epson sequence, the number of whole 256 dot-column units contained in the total number (N) of dot columns. And like the Epson's $N1$, the TRS-80's $N2$ is the "leftover" part of the total, that is, the total minus the product of $N1$ and 256. To convert any total (N) to values of $N1$ and $N2$ for positioning, then, use the following formulas:

$$N1 = \text{FIX}(N/256) : N2 = \text{FIX}(N - 256 * N1)$$

Figure 3-4 demonstrates how the position of a single dot print changes as the N is increased from 1 to 300 in single steps over 300 successive line feeds of $1/72$ inch (program 3-2). The top line in the figure shows what happens with a standard (pica) print density. If you have a TRS-80 system for this program to run on, you'll see the values of N , $N1$, and $N2$ parading on the screen as the printer steps up the page; notice that as N passes from 255 to 256, $N1$ changes from 0 to 1 and $N2$ drops from 255 to 0.

How big a distance the printhead travels in carrying out a direct-positioning command depends on the print density as well as the value of N . The lower two lines of figure 3-4 show this. These were drawn by the printer (a DMP-400) from the same starting point on the paper as the pica-density line at the top, after just manually reeling the paper back to where a pencil mark was made before the printer was run.



Comparing all three lines shows a clear difference among the three densities in how far the printhead travels horizontally. For better comparison program 3-2 switches dot codes at the 55th, 155th, and 255th values of N within the series of 300 increments. To travel all the way across an eight-inch page, N would have to keep on incrementing to a value of 480 for pica, 560 for elite, and 800 for condensed pitch. On the fifteen-inch DMP-400 (which has line lengths of 13.2 inches), these N values become 792, 950, and 1,320.

PROGRAM 3-2. POSITION/TRS. Horizontal positioning using the TRS-80's $27 + 16 + N1 + N2$ command.

```

10 'PROGRAM 3-2: "POSITION/TRS" -- DIRECT-POSITIONING ON TRS-80 PRINTERS
20 'FOR TRS-80 MODEL III, REQUIRES BOOTHE'S PRINTER DRIVER (PROGRAM 3-5)
30 PRINT "SELECT ONE OF THE FOLLOWING PRINT DENSITIES:"
40 INPUT " 1. PICA 2. ELITE 3. CONDENSED (TYPE 1, 2, OR 3)";PD
50 ON PD GOSUB 1000, 2000, 3000
60 LPRINT CHR$(18);
100 FOR N=0 TO 300
110 N1=FIX(N/256); N2=FIX(N-256*N1)
120 PRINT "N=";N,"N1=";N1,"N2=";N2
130 IF N=55 OR N=155 OR N=255 THEN DC=255 ELSE DC=136
140 LPRINT CHR$(27);CHR$(16);CHR$(N1);CHR$(N2);CHR$(DC);

```



```

150  LPRINT CHR$(27);CHR$(50);           '1/72" LINE FEED
160  NEXT N
170  END
1000 LPRINT CHR$(30);CHR$(27);CHR$(19);: RETURN      'PICA
2000 LPRINT CHR$(30);CHR$(27);CHR$(23);: RETURN      'ELITE
3000 LPRINT CHR$(30);CHR$(27);CHR$(20);: RETURN      'CONDENSED

```

If you do a lot of programming with the direct-positioning command, you quickly learn to cut down your labor by defining shorthand strings to take the place of `CHR$(27);CHR$(16)` in the `27 + 16 + N1 + N2` sequence. For example, a string called `PS$` (for “positioning string”) could be defined by the statement `PS$=CHR$(27)+CHR$(16)` early in a program, and every time the positioning sequence is needed after that, the print command could start with `LPRINT PS$;CHR$(N1);CHR$(N2)`.

This idea can be extended. For example, when you know that all `N` values will be under 256 and therefore all `N1` values will be 0, you could use a longer string defined by `PO$=CHR$(27)+CHR$(16)+CHR$(0)` for insertion into `LPRINT PO$;CHR$(N2)` statements.

The Radio Shack positioning command is efficient, too. It is not necessary for the printhead to return to the left margin during the execution of the command—it just goes directly from wherever it is to the new location that is specified by `N1` and `N2`. With this fact combined with bidirectional printing, the positioning is fast as well as efficient.

The Apple Dot Matrix and Imagewriter printers also have a direct-positioning command—the `27 + F + nnnn` sequence. In actual practice, `nnnn` would never be higher than 1280, the number of dot columns in a print line with a density of 160 dots per inch.

On the IBM and Epson printers, even the FX-80, the nearest thing to this direct-positioning feature you can find is the horizontal tabbing command `27 + D + N1 + N2 + ... + 0`. That can move the printhead to every sixth dot column in standard density, every twelfth in double density, or every twenty-fourth in quadruple density. From there on, if you want it to get to some in-between position, you have to combine this command with some other kind of command. We’ll see shortly that, as usual, there is more than one way to solve this problem.

Repeat-blank positioning. With just about any printer, the printhead automatically moves—one dot column—every time it prints a dot pattern in the bit-graphics mode. (In text mode, except for proportional printing, it moves six dot columns after every character print.) In both modes, this printhead movement will take place even if the pattern or character is a blank. In text mode, `CHR$(32)` is a universal code for a blank space that is six dot columns wide. With IBM, Epson, and Apple printers in graphic mode, the Microsoft BASIC command `LPRINT CHR$(0)` will result in a blank dot-column movement, as will `LPRINT CHR$(128)` on the Tandy printers.

We can cash in on these facts to develop a new way of positioning the printhead. For those printers lacking the direct-positioning command, in fact, this turns out to be the main way and sometimes the only way that makes any sense. Repeating a blank code N times will move the printhead to the right N dot columns in graphics mode and $6 \times N$ dot columns in text mode. This means that if you want to move the printhead from the left margin to the 200th dot column in graphics mode, use `LPRINT STRING$(200,0)` on the IBM, Epson, and Apple printers and `LPRINT STRING$(200,128)` on the TRS-80 printers.

If your computer's BASIC doesn't have the `STRING$` function, use a repeat code that the printer's ROM responds to, such as the `CHR$(27) + "V0200" + CHR$(0)` sequence for the Apple or `LPRINT CHR$(28);CHR$(200);CHR$(128)` for the TRS-80s. If you lack both `STRING$` on the computer and a repeat command on the printer, then your only choice for repeating is a `FOR-NEXT` loop. A `STRING$`-less Apple computer driving a nonrepeating Epson printer, for example, would have to use `PR#1: FOR N=1 TO 200: PRINT CHR$(0);: NEXT N`.

Repeat-blank positioning can move the printhead to the right from any position in a print line, not just the left margin. If you've already gotten to the 100th dot column (by whatever means) and you want to move to the 120th before printing any more dot patterns, then just insert twenty blanks. This means that repeat-blank positioning can be combined with horizontal tabbing, direct positioning, other repeated blanks, or positions resulting from the printing of characters

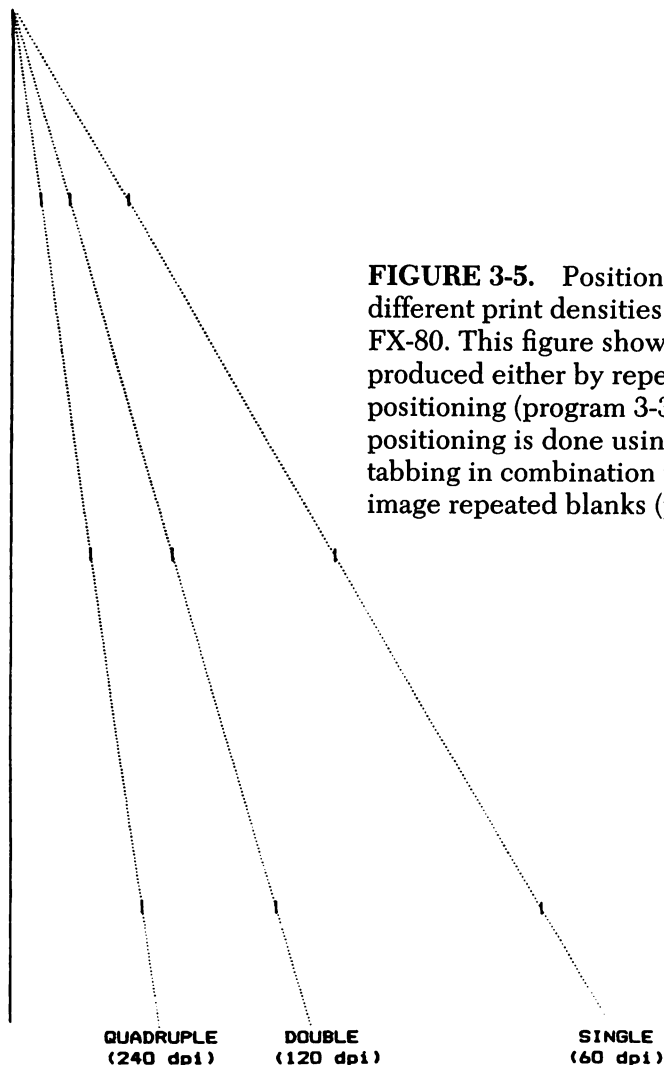


FIGURE 3-5. Positioning with three different print densities on the Epson FX-80. This figure shows the printout produced either by repeat-blank positioning (program 3-3) or when positioning is done using text-mode tabbing in combination with bit-image repeated blanks (program 3-4).

or dot patterns earlier in a print line to address any dot column.

Figure 3-5 shows the result of using repeat-blank positioning at three different densities with the Epson FX-80 printer. As with figure 3-4, the printout was run three times from the same starting point on the paper, and code 255 was used to mark the 55th, 155th, and 255th positions in the sequence of 300. At each density a dot was positioned by starting at the left margin and sending a series of blank codes (e.g., for the 78th position using `STRING$(78,0)` or for the 293rd position using `STRING$(255,0)STRING$(38,0)`) before printing the dot.

This being repeat-blank positioning in the bit-image mode, the repetition of blanks and printing of the dot had to be preceded by a graphic reservation command that specified the number of dot columns for the blanks, in addition to a reservation of one dot column for the printed dot. Be careful in specifying the N1 and N2 values for the reservation of blank dot columns and the values inside the parentheses of STRING\$(), because each unit of N2 in the 27+"K"+N1+N2 command represents 256 dot columns (including dot-column 0 at the left margin), whereas the limit for repeating any code in the STRING\$() function is 255. This gets to be a problem as soon as you pass the 255th dot column, when N2 changes from 0 to 1.

Continuing our example of the 293rd position, the full command for reserving dot columns for blanks alone takes the form `LPRINT CHR$(27)"K"CHR$(37)CHR$(1)` since 293 minus 256 equals 27. But the combination of STRING\$() functions for moving the printhead over 283 blank positions (including dot-column 0) is `STRING$(255,0)STRING$(38,0)`, since 293 minus 255 (the limit) is 38. If we were above the 512th position (in double density), we would have to add 2 rather than 1 to the value of N1 in the final STRING\$() specification. For every multiple of 256, we need to add a value of 1 in the number of repetitions specified in the sequence of STRING\$() functions. Fortunately, multiples of 256 are given by N2, so we can arrange for this incrementing to take place automatically in the more general case by inserting `STRING$(N2,0)` before the other STRING\$() provisions.

If we are limited to 480 dot columns (numbered 0 to 479) in single density and we want to place a dot in each column, we start by varying N (the dot-column address) from 0 to 479 in a FOR-NEXT loop and within the loop converting each N value to N1 and N2 values in the usual manner, using the general formulas $N2 = \text{FIX}(N/256)$ and $N1 = \text{FIX}(N - 256*N2)$. Then, continuing inside the loop, the graphic space reservation and positioning command can be `LPRINT CHR$(27)"K"CHR$(N1)CHR$(N2)STRING$(N2,0)STRING$(255*N2,0)STRING$(N1,0);`. Finally, with a separate reservation in another statement, the dot is printed by `LPRINT CHR$(27)"K"CHR$(1)CHR$(0)CHR$(8)`.

With more than 480 dot columns, the easiest way to repeat blanks is with a simple counter loop, for example (with

double density), FOR K=1 TO N: LPRINT CHR\$(27)"L" CHR\$(1)CHR\$(0)CHR\$(8);: NEXT K. For that matter, a FOR-NEXT counter is the simplest way for any number of dot columns, but it is surprisingly slower than a series of STRING\$() functions when N is over 500. In a program for drawing a graph in which several dozen data points need to be positioned, this time difference can amount to many minutes.

PROGRAM 3-3. REPBLANK/EPS. Positioning on the Epson FX-80 with repeated blanks.

```

10 'PROGRAM 3-3: "REPBLANK/EPS" -- REPEAT-BLANK POSITIONING ON THE EPSON FX-80
20 'FOR TRS-80 MODEL III (WITH BOOTHE'S DRIVER) AND FX-80 IN QUADRUPLE DENSITY
30 CLEAR 2000
40 LPRINT CHR$(27)"@";
50 LPRINT CHR$(27)"3"CHR$(1);
100 FOR N=0 TO 1919 STEP 20
110   N2=FIX(N/256): N1=FIX(N-256*N2)
120   PRINT "N=";N,"N1=";N1,"N2=";N2
130   FOR K=0 TO N2-1: S(K)=255: NEXT K
140   FOR K=N2 TO 7: S(K)=0: NEXT K
150   FOR K=0 TO 7: PRINT S(K);: NEXT K: PRINT
160   LPRINT CHR$(27)"Z"CHR$(N1)CHR$(N2)STRING$(N2,0)STRING$(S(0),0)STRING$(S(1)
      ,0)STRING$(S(2),0)STRING$(S(3),0)STRING$(S(4),0)STRING$(S(5),0)STRING$(S(6)
      ,0)STRING$(S(7),0);STRING$(N1,0);
170   LPRINT CHR$(27)"K"CHR$(1)CHR$(0)CHR$(8);           'PRINT DOT
180   LPRINT CHR$(10);
190 NEXT N
200 END

```

Program 3-3, called REPBLANK/EPS, is designed for rapid positioning over dot columns 0 through 1919 when the FX-80's quadruple density is used. The key feature of this program, in line 160, is a chain of ten STRING\$() functions following the usual graphic reservation. The first of the ten, STRING\$(N2,0), is the provision just noted—a blank print for every multiple of 256 above dot-column 255. The last of the ten, STRING\$(N1,0), makes up the difference between N and the next lowest even multiple of 256, as before. Between the first and last are eight STRING\$() functions, each of which can take the values of STRING\$(0,0) or STRING\$(255,0).

Which of these two values that any of these middle eight

STRING\$() functions has is determined by storing values of 0 or 255 in an array called S(K). Lines 130 and 140 of REPBANK/EPS assign these values with a pair of FOR-NEXT loops having lengths determined by the number of even multiples of 256, that is, by N2. When S(K) equals 0, its corresponding STRING\$(S(K),0) function takes the form STRING\$(0,0), meaning no blanks and no movement of the printhead; when S(K) equals 255, the printhead will move 255 dot columns as a result of STRING\$(255,0). As N increases, the number of middle-eight STRING\$() functions containing 255 instead of 0 will increase. If enough string space is arranged ahead of time by the CLEAR statement in line 30, the principle here can be extended for printers having more than 1,920 dot columns.

Compared to simple FOR-NEXT loops for repeating blanks, program 3-3's more complicated method pays off in time saved. With the TRS-80 Model III driving the FX-80 from dot-columns 0 through 1919 in steps of 20, the FOR-NEXT method takes 12.33 minutes to get across the page. With program 3-3 that time is reduced to 3.08 minutes, or one fourth as long.

Program 3-4 presents an alternative method for positioning with printers such as the IBM and Epson models that have neither the direct-positioning feature nor the repeat-character feature. With this method, horizontal tabbing, using TAB() in text mode, is combined with repeated blank codes, using CHR\$(0) in graphics mode. The program varies dot-column positions (DC) from 1 to 300 (line 100) and calculates the number of tab stops (P1) by dividing DC by 6 (line 110). This is because when these printers are set to single-density pitch, there are six dot columns from one tab stop to the next. The P1 value is used for tabbing to the nearest tab stop below the target position (DC) and then filling in the remaining distance by repeating up to six blanks. This remaining distance, represented by LO, is equal to the number of leftover bit-image steps when the position of nearest tab stop is subtracted from the target position; hence $LO = FIX(DC - 6 * P1)$ in line 120. After the tabbing portion of the positioning is executed (line 160) in text mode, a graphic reservation of LO dot-columns is made and the repeat-blank portion of the positioning is accomplished with STRING\$(LO,0) in line 170. Finally, with one more column

PROGRAM 3-4. POSITION/EPS. Tabbing combined with repeated blanks in horizontal positioning on the Epson FX-80.

```

10 'PROGRAM 3-4: "POSITION/EPS" -- HORIZONTAL POSITIONING ON THE EPSON FX-80
20 'FOR TRS-80 MODEL III, REQUIRES BOOTHE'S PRINTER DRIVER
30 LPRINT CHR$(27)"a";           'MASTER RESET
40 LPRINT CHR$(27)"3"CHR$(2);    'SET 2/216" LINE FEED
50 LPRINT CHR$(13);              'CARRIAGE RETURN
100 FOR DC=1 TO 300
110   P1=FIX(DC/6)                'P1=NUMBER OF TAB STOPS
120   LO=FIX(DC-6*P1)             'LO=LEFT-OVER BIT-IMAGE STEPS
130   P1=P1+1                    'ADJUST TAB STOPS TO START AT 1
140   PRINT "DC=";DC,"P1=";P1,"LO=";LO
150   IF DC=55 OR DC=155 OR DC=255 THEN CC=255 ELSE CC=8
160   LPRINT TAB(P1);             'TAB PORTION OF POSITIONING
170   LPRINT CHR$(27)"K"CHR$(LO)CHR$(0);STRING$(LO,0); 'REPEAT-BLANK PORTION
180   LPRINT CHR$(27)"K"CHR$(1)CHR$(0);CHR$(CC);      'PRINT OF DOT OR SLASH
185   REM -- CHANGE "K" TO "L" FOR DOUBLE DENSITY, TO "Z" FOR QUADRUPLE DENSITY
      IN LINES 170 AND 180
190   LPRINT CHR$(10);           'LINE FEED
200 NEXT DC
210 END

```

reserved, the dot is printed (line 180). An alternative would be to delete line 180 and change line 170 to

```

      LPRINT CHR$(27)"K"CHR$(LO+1)CHR$(0)
      STRING$(LO,0)CHR$(CC)

```

Figure 3-5 portrays the result of running program 3-4 at three different densities on the FX-80 just as well as it did in showing printouts from positioning done entirely by repeated blanks. If you're combining repeated blanks with tabbing at double density, there will be twelve dot columns between tab stops, and at quadruple density there will be twenty-four. The denominator in line 110 should be changed from 6 to 12 or 24 along with the changes from "K" to "L" or "Z" in lines 170 and 180.

Generally it'll be simpler to stick to repeat-blank positioning than to combine that with horizontal tabbing, especially if your computer's BASIC has the STRING\$ function. If you have a lot of target positions falling right at tab stops, such as numbers or hash marks on the axis of a graph, then tabbing might be simpler. Also, if you have to move distances greater than 255 dot-columns (the limit of a single STRING\$) very often, tabbing combined with repeated blanks may be just as convenient as program 3-3's repeated STRING\$ method.

The proportional spacing command. The Tandy and Apple (but again, not the IBM and Epson) printers have what is called the proportional space command. This uses the code sequence `CHR$(27);CHR$(N)` and moves the printhead N dot columns to the right from any position in a line. N can vary from 0 to 9 on the TRS-80s and from 1 to 6 on the Apple printers. In both cases, the command can only be executed in text mode. Indeed, it was designed for adding tiny spaces between words in order to make all nonindented lines of text the same length (right justification).

But the proportional spacing feature also provides yet another way of making more use of horizontal tabbing features for positioning in graphics. It is most useful in graphics at pica (text-mode) and single-density (graphic-mode) pitches. For moving the printhead to dot-column addresses that are between tab stops, you can combine a tabbing command with a proportional-space command the same way that tabbing was combined with repeated blanks in figure 3-6, except for staying in text mode all the way to the target position. To reach the sixty-fourth dot column, for example, you could use `LPRINT TAB(10);CHR$(27);CHR$(4)` and then go into graphic mode for some bit-image printing.

At times this can be a handy way of getting the printhead to where you want it. And for Apple users there is the advantage that the graphic reservation command (the 27+“G”+nnnn sequence) that accompanies the positioning doesn't have to include a few blanks that make up part of the positioning.

A word of warning, though: The distance the printhead moves in proportional spacing may not be the same as the movement in repeat-blank positioning. For instance, the command `CHR$(27);CHR$(6)` may produce a movement of only three addressable dot columns instead of six. This needs to be checked (and if necessary, calibrated) for your particular printer.

Backspacing. The universal backspacing code `CHR$(8)` has some uses in positioning in graphics, at least occasionally. This is when it takes the form `CHR$(8);CHR$(N)`, letting you set the value of N, the number of half dot columns. (For every two steps in N, the printhead moves one dot column to the

left.) The Radio Shack printers have this feature but the IBM-Epson and Apple printers do not—their backspace is limited to the width of a text character. Since backspacing seems to have been designed for text strikeouts, such as combining 0 and / into Ø, this command is available only in text mode, even with the TRS-80 printers.

The backspacing command gets a lot of use with these printers in placing data points in graphs. Data-point symbols, such as circles, diamonds, and triangles, are usually five to seven dot spaces in height, as we'll see in chapter 9. We usually place data points on a graph using the direct-positioning command (the $27 + 16 + N1 + N2$ sequence), but if that is all we use, it will be the bottom of the data point, not its center, that represents the numerical value on the graph. Since we want the center of the data point to represent the value, we need to start printing it a bit lower—half the height of the data point lower, in fact. Inserting a $30 + 8 + N + 18$ sequence just after the $27 + 16 + N1 + N2$ will remedy this difficulty. If, for example, your data point is seven dots high, an N of 7 in the $30 + 8 + N + 18$ sequence will drop the printing $3\frac{1}{2}$ dot columns, and now the center of the data-point symbol will be right where you want it.

Printers not having the $8 + N$ type of backspacing, i.e., those that automatically backspace six dot columns when they receive `CHR$(8)`, won't be able to fine-tune data-point placements the way the $8 + N$ printers do. They are forced to other ways of positioning, such as reducing the number of dot columns the printhead moves in repeat-blank positioning, or combining the six-dot backspace with small repeat-blank adjustments.

There are other uses of backspacing in bit-image graphics that will show up in later sections of this book. In the very next section, in fact, comes another important use of the $8 + N$ form of backspacing.

Trouble Codes

If there is any topic that may be more crucial to dot-matrix printer graphics than positioning, it is dealing with trouble codes. There are many codes which, when sent to a printer in graphic mode, cause outrageous events—unwanted form feeds, printing in the wrong position, failure to repeat dot

```

▪ 0
▪ 1
▪ 2
▪ 3
▪ 4
▪ 5
▪ 6
▪ 7
▪ 8
▪ 9
▪ 10
▪ 11

```

FIGURE 3-6. Trouble codes: incorrect positioning with codes 10, 12, and 96 with the TRS-80 Model III driving a DMP-400 or LPVIII printer.

(FORM FEED FROM CODE 12)

```

▪ 13
▪ 14
▪ 15
▪ 16
▪ 17
▪ 18
▪ 19
▪ 20

```

```

▪ 90
▪ 91
▪ 92
▪ 93
▪ 94
▪ 95
▪ 96
▪ 97
▪ 98
▪ 99
▪ 100

```

patterns or blanks correctly, and other sources of exasperation. Almost every system has at least one of these trouble codes. This makes one wonder if perhaps there have been ten thousand would-be programmers trying to draw graphs or pictures on dot-matrix printers who have given up as a result of trouble codes they didn't know even existed, let alone how to detour around them.

Some examples. You can tell from the Epson printer manuals that a big trouble-code area has been in systems in which MX-80 and FX-80 printers are driven by the TRS-80

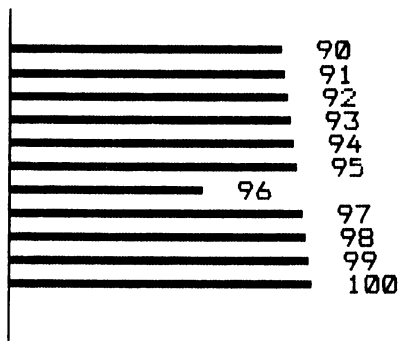
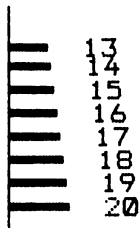
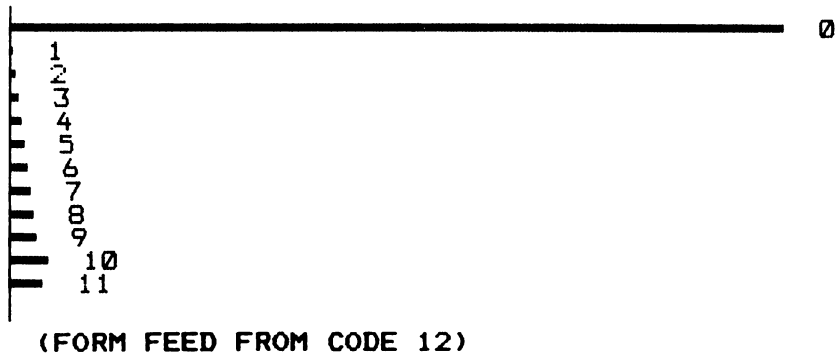


FIGURE 3-7. When the `CHR$(28);CHR$(N)` sequence is used for repeating a dot code with the TRS-80 Model III, errors in repeating may show up when `N` is 0, 10, 12, or 96.

Model I and Model III computers. But even more difficulties show up when these computers are driving TRS-80 printers! Figure 3-6 shows mispositioning and unwanted form feeds from codes 10, 12, and 96 on the Radio Shack DMP-400 and LPVIII printers when they are interfaced to a Model III. When values of `N2` in the direct-positioning sequence `27 + 16 + N1 + N2` are incremented from 1 to 100 and sent one at a time, the 10th value starts printing in the 13th dot column

instead of the 10th, the 12th value produces a form feed of several inches, and the 96th value prints at the 64th dot column!

Figure 3-7 shows that these same codes cause misbehaviors in repeat-character printing. When the single-dot print of dot code 136 is repeated from 0 to 100 times using the 28-N command, there are 13 repetitions instead of 10, 64 instead of 96, 255 instead of 0, and (again) a form feed from code 12. And the same troubles show up when you use repeat-blank ($28 + N + 128$ instead of $28 + N + 136$) positioning.

Don't blame the printers for these problems. By and large, the Radio Shack and other brands of printers faithfully print the patterns of the whole range of dot codes that they receive; it is in the control codes where most of the problems arise. The same combinations of computers and printers that have trouble repeating a single dot print with N's of 0, 10, 12, and 96 in the $28 + N$ command show no difficulties with the commands `STRING$(0,136,)`, `STRING$(10,136)`, `STRING$(12,136)`, and `STRING$(96,136)`, or when they use the `STRING$` function to execute repeated blanks. Furthermore, the problems vanish when the computer's print driver is changed, if the `PRINT#` or `OUT` commands are used instead of `LPRINT`, or (with the TRS-80 Models I and III and Apple II computers) if a `PEEK-and-POKE` statement is substituted for `LPRINT` or `PRINT`.

So trouble codes are software, not hardware, problems, which almost always arise from the computer rather than the printer. Some of them are partly understandable when you remember that `CHR$(12)` is a universal form-feed code, or you realize that the Model III automatically converts `CHR$(10)` to `CHR$(13)` to produce carriage returns along with line feeds. But these things aren't supposed to happen in graphic mode!

Other TRS-80 computers have trouble codes of different sorts or numbers. The Model I can't send codes 0, 10, 11, or 12 by `LPRINT` commands. The Model 100 and Model 4 (with the TRSDOS 6.0 operating system) have trouble with code 9—with the DMP-400, code 9 behaves like code 32 in direct positioning and like a true zero (not 255) in the $28 + N$ repeat sequence. When the Model 4 is run in Model III mode, there are no problems except with some of earliest Model 4s sold.

The IBM PC follows `LPRINT CHR$(13)` with `CHR$(10)`

automatically in text mode, so that carriage returns will be accompanied by line feeds, but unfortunately this also happens in graphic mode when you're trying to use CHR\$(13) as a dot code. The result can be gibberish interrupting a sequence of dot patterns or several feet of blank paper, depending on which printer is cabled to the PC.

With Apple II computers using Applesoft BASIC's PRINT command (after PR#1), the problems are not so much with individual trouble codes as with whole banks of codes. With the Apple II+, the printer won't receive codes in the range of 128 to 255, only the seven-bit codes from 0 to 127 (although some printers, such as the Epson FX-80, have software commands that enable you to use the 128-to-255 range or the 0-to-127 range but not both at the same time). The Apple IIe, on the other hand, sends only eight-bit codes in the range of 128 to 255, at least when it is interfaced to the Imagewriter with a Super Serial Card. This means that the eighth bit, corresponding to the printhead's bottom pin (the 128 pin), is always set (fired) by Applesoft, the result being that all dot codes below 128 will be printed with an unwanted "underline." And if you try to make the bottom pin stop printing all the time with a software command for ignoring the eighth bit, the system will simply hang up. Several additional problems with an Apple II+ driving an MX-80 or FX-80 are discussed in Epson's manuals for these printers. Compared to these Apple problems, the TRS-80 and IBM difficulties seem like a picnic with only a few ants.

Diagnostic aids. One way of detecting problem codes, already illustrated in figures 3-6 and 3-7, is to expose your printer to the whole range of control codes and dot codes it would ever receive. If you do this systematically, such as using regular increments in the dot codes being tested, the trouble codes will usually stick out like sore thumbs in the printouts. It's a good idea to test the same sets of codes serving in different roles, i.e., in positioning, in repeat printing, in reserving graphic space, and as dot codes.

A quicker way of testing every code for possible trouble is a "hex dump." The Epson FX-80 is one of the few printers having this feature, but probably many other printer manufacturers will add it in the near future. If you hold down the FF (form feed) button as you are turning on the FX-80, you

TABLE 3-2
Hex Dump for the Case of No Trouble Codes

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B
3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63
64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77
78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86	87	88	89	8A	8B
8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD
BE	BF	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1
D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5
E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	FO	F1	F2	F3	F4	F5	F6	F7	F8	F9
FA	FB	FC	FD	FE	FF	OD													

will put it in “hex mode,” meaning that it will convert all decimal codes and print them as hexadecimal codes. This will tell you just what codes are being received by the printer, as opposed to those that your BASIC program is trying to send to the printer.

Sending the whole range of decimal codes (0 to 255) can be done by entering **FOR I=0 TO 255: LPRINT CHR\$(I);: NEXT I**. With Apple II computers, add **PR#1** before the **FOR-NEXT** loop and use **PRINT** instead of **LPRINT** inside the loop. If there are no trouble codes, the printout should look like table 3-2.

The same test with an IBM PC driving the Epson FX-80 will show a very similar printout but reveal an 0A (decimal 10) right after 0D (decimal 13) instead of continuing right on to 0E (decimal 14). This is perfectly in line with the fact that a line feed routinely follows a carriage return when the print command is **LPRINT** (but not when it is **PRINT#**—see below).

The TRS-80 Model III's case looks far worse (table 3-3), with decimal 10 (0A) converted to 13 (0D) and a long string of consecutive 0A's in place of codes 12 and 13 (0C and 0D). No wonder, then, that in figure 3-7 the dots ended up in position 13 instead of 10, and sending code 12 produced a long series of line feeds. Incidentally, the hex dump shows nothing abnormal in the vicinity of code 96, however, so perhaps the displacement to position 64 is a chip flaw in some of the early TRS-80 printers.

Table 3-4 shows a hex dump with the TRS-80 Model 4 and Model 100. The trouble with code 9 shows up clearly—a

TABLE 3-3
Hex Dump for the TRS-80 Model III Computer

00	01	02	03	04	05	06	07	08	09	0D	0B	0A	0A	0A	0A	0A	0A	0A	0A
0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A
0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A
0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0A	0E	0F	10
11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24
25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38
39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A	4B	4C
4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60
61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74
75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86	87	88
89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	9C
9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	AB	A9	AA	AB	AC	AD	AE	AF	BO
B1	B2	B3	B4	B5	B6	B7	BB	B9	BA	BB	BC	BD	BE	BF	CO	C1	C2	C3	C4
C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	DO	D1	D2	D3	D4	D5	D6	D7	DB
D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC
ED	EE	EF	FO	F1	F2	F3	F4	F5	F6	F7	FB	F9	FA	FB	FC	FD	FE	FF	OD

series of eight 20s (decimal 32s). So positioning at 32 instead of 9 with these computers is somewhat understandable, but the failure to repeat-print at all when they send the sequence `CHR$(28);CHR$(9)` is not. The FX-80 misbehaves like a child when code 9 is delivered to it by these computers in either text or graphics mode. If you need to use this code to activate horizontal tabbing, add 128 to it to get `CHR$(137)` and there should be no problem in text mode. In graphics mode, you'll have to make a detour around code 9 (see below).

When the newer and supposedly improved TRSDOS 6.2 operating system is used with the Model 4 instead of its original TRSDOS 6.0 system, a hex dump reveals what may be a new record for the number of trouble codes—not only code 9, but codes 10, 13, 112, and 184 as well!

Many computers, including the IBM PC, Apple IIs, and Epson HX-20, automatically add and send carriage-return (0D) and line-feed (0A) codes at regular intervals (e.g., every 80th code) during a stream of codes from a BASIC program. Obviously these are unwanted in graphic printing, but nonetheless they still are sent by the computer and printed in graphic mode as dot codes 13 and 10, interrupting whatever else you were trying to print. This can be revealed by a simple repeat command such as `LPRINT STRING$(245,DC)`. As we'll see, there's a ready cure for this ailment provided by those BASICs that have the `WIDTH` statement.

TABLE 3-4
Hex Dump for the TRS-80 Models 4 and 100

00	01	02	03	04	05	06	07	08	20	20	20	20	20	20	20	0A	0B	0C	
0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1B	1C	1D	1E	1F	20	21
22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35
36	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49
4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D
5E	5F	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71
72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85
86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99
9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD
AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	C0	C1
C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	DO	D1	D2	D3	D4	D5
D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9
EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD
FE	FF	0D																	

It's a good bet that nearly every microcomputer has at least one trouble code. This review doesn't cover them all.

Remedies. The good news is that there are solutions to almost any of the problems generated by trouble codes. The remedies take several different forms, and they have to be tailored to the computer or computer-printer combination. The alternatives include using more direct output commands than LPRINT such as POKE, OUT, or PRINT#, changing the computer's printer driver, extending WIDTH, and introducing detours in the BASIC programs.

There's more to the LPRINT command than meets the eye. To the computer it is several instructions, including such things as converting certain codes to produce line feeds on the printer and checking to make sure the printer is ready to receive data before any are sent to it.

Some of our problems with certain codes can be avoided if we bypass the LPRINT command and use simpler, more direct commands. For years now, TRS-80 Model I users have solved the problem with codes 0, 10, 11, and 12 by abandoning LPRINT in favor of the more direct POKE command. POKE will send any code to the printer without being intercepted, converted, or coupled with another code.

Unfortunately, it will send the code even when the printer is not ready to receive one. To avoid losing data from this, a PEEK at the printer's ready-or-not status must come before the POKE. The usual form for the Model I is:

```
100 IF PEEK(14312) <> 63 THEN GOTO 100 ELSE
    POKE 14312,DC
```


where 14312 is the RAM address for the printer's status and outgoing codes, 63 the printer's indication of "ready to receive," and DC is a dot code to be sent.

With the TRS-80 Model III, the Model I's PEEK-and-POKE solution can be used, or an analogous INP-and-OUT method suggested by David Lien in Epson's MX-80 manual:

```
100 IF INP(251) <> 61 THEN GOTO 100 ELSE OUT
    251,DC
```

But Bob Boothe (in *80 Micro*, March 1982) has come up with an even better solution for both models—changing their printer drivers. For the Model I his subroutine for doing this is given in program 3-5.

PROGRAM 3-5. PRNTDRVR. Boothe's printer driver for the TRS-80 Model I.

```
30000 'PROGRAM 3-5: "PRNTDRVR/SUB",A -- BOOTHE'S PRINTER DRIVER FOR MODEL I
30005 'Presented by Bob Boothe (80 Micro, March 1982)
30010 DATA 21E837CB7E20FC211100397ED3FBC9
30020 READ Z$: V=16571
30030 FOR W=1 TO LEN(Z$) STEP 2
30040   Z=ASC(MID$(Z$,W,1))-48
30050   IF Z>9 THEN Z=Z-7
30060   U=ASC(MID$(Z$,W+1,1))-48
30070   IF U>9 THEN U=U-7
30080   POKE V,Z*16+U
30090   V=V+1
30100 NEXT W
30110 POKE 16422,187
30120 POKE 16423,64
30199 RETURN
```

For the Model III only one small change is needed: substitute 32E387 for D3FB near the end of line 30010 of the Model I program. Once the subroutine is loaded (while you're in BASIC) and inserted (use **GOSUB 30000**), it doesn't need to be repeated for any additional programs in the session unless you turn the computer off. With the printer driver thus changed, you can use **LPRINT** to your heart's content. And remember to thank David Kater's Epson FX-80 manual for presenting this solution, too.

We have no programs to offer here for changing printer drivers on Apple computers, but still another hero, Jerry Bialzik of the electronics shop in the Psychology Department of the University of Wisconsin at Madison, has come to the

rescue of Apple IIe users. Confronted with the dual problem of the constant printing by the 128 pin with dot codes under 128 and phantom prints before and after a programmed sequence of dot patterns, he took little more than five minutes to find the IIe's memory addresses for detecting the Imagewriter's ready signal and for sending it data, and came up with the following solution:

```
100 PR#1
110 IF PEEK(49305) <> 16 THEN GOTO 110
120 POKE 49304,DC
```

With this alternative to the PRINT command, both problems vanished—the whole range of DCs from 1 to 255 printed dot patterns correctly, and there were no phantom prints. In certain Applesoft programs on the IIe, however, unwanted prints before a run of dot patterns may still show up, but they can apparently be eliminated by placing **PRINT CHR\$(24)**; before the PEEK and POKE statements, which clears the printer's buffer. (Program 6-4 is a case in point.) As with the TRS-80 Models I and III before Boothe's driver change came into use, the PEEK-and-POKE method may be the only reliable way of sending dot codes in bit-image graphics with this system. Sending control codes with PRINT seems to pose no problems, however.

Kater (in the Epson FX-80 manual) had already advocated this solution for the Apple II+, except the address for PEEK is 49601 and for POKE is 49296. For the IIc, the addresses are the same as for the IIe, assuming the Super Serial interface is used with the Imagewriter.

Still another solution, most applicable with computers having a strong BASIC such as the IBM PC, uses PRINT# instead of LPRINT. To avoid unwanted line feeds with carriage returns and periodic carriage returns interrupting print lines, PRINT# has to be used following OPEN and WIDTH statements. For the IBM, the proper form is:

```
100 OPEN "LPT1:" AS #1
110 WIDTH #1, 255
120 PRINT#1,DC
```

where LPT1 refers to the line printer, #1 to a buffer, and DC to a dot code. The middle line sets the printer's width to "infinity" (the highest value possible), to prevent carriage

returns in the midst of print lines. With PRINT#1 instead of LPRINT, a line feed will no longer follow carriage returns when they are programmed to occur.

With other systems, most of which probably won't have the PRINT# alternative, you'd be wise to try PEEK-and-POKE first, and if that doesn't work, or you can't find the addresses, you can fish around for INP and OUT ports (there are only 255 ports to search through on many computers). If these solutions fail, you can hope that a real programmer will publish an improved printer driver for your system, or use detours.

Detours in BASIC. The last-resort strategy is not to get rid of the trouble codes, but to detour around them with some small changes in your BASIC program. Sometimes simply incrementing or decrementing a trouble code whenever it appears will work, in the sense of producing such a tiny change in the position of a print that it is unnoticeable. This usually won't hurt if your bit-image printing is done in the higher densities. On the TRS-80 Models 4 and 100, for example, the problems with CHR\$(9) can be dodged by inserting this detour into the program:

```
100 IF N2=9 THEN N2=8
```

On the Model III (if Boothe's printer-driver change were not a better way), the detour might take the form:

```
100 IF N2=10 OR N2=12 OR N2=96 THEN N2=N2+1
```

If the trouble code is intended to be a dot-pattern code instead of a positioning code (direct or repeat-blank), then replacing it with another code having a similar pattern makes more sense than incrementing, for example, IF DC=12 THEN DC=8.

If you're determined to get a dot print in exactly the correct position, you can increment and then backspace. With half-dot-column backspacing on the TRS-80s, two backspaces will correct for increasing the value of the direct-positioning code by one. This kind of detour takes the following form:

```
100 BS$=CHR$(30)+CHR$(8)+CHR$(2)+CHR$(18)
    'BACKSPACE STRING
```

```

110 IF N2=10 OR N2=12 OR N2=96 THEN Z=N2+1:
    LPRINT CHR$(27);CHR$(16);CHR$(N1);CHR$(Z);
    BS$;CHR$(DC);:GOTO 130
120 LPRINT CHR$(27);CHR$(16);CHR$(N1);
    CHR$(N2);CHR$(DC);
130 (program continues)

```

Notice that a new variable, Z, is introduced here when the trouble codes (10, 12, or 96) occur, and it is Z that is set equal to N2+1 instead of N2 itself being raised by one. That leaves the original values of N2 intact in case those values are needed elsewhere in the program. The detour positioning command (line 110) uses Z as N2 in its 27+16+N1+N2 sequence together with the backspace correction defined in line 100. The combination of forward and backward movement of the printhead puts the dot code (DC) right where you want it for the utmost accuracy.

Tandy's 8+N form of backspace can also be used in dealing with trouble codes in repeat printing, but that depends to some extent on what kind of dot pattern is being repeated. Usually it'll be better to decrease rather than increase the number of repetitions, and then add a single additional print of the desired dot pattern (or blank) to make up the difference, instead of incrementing and then backspacing. When using the 28+N repeat command, for example, the detour could take this form:

```

100 IF N=10 OR N=12 OR N=96 THEN LPRINT
    CHR$(28);CHR$(N-1);CHR$(DC);:
    GOTO 120
110 LPRINT CHR$(28);CHR$(N);CHR$(DC);
120 (program continues)

```

But watch out: This detour won't work on the Radio Shack printers if the trouble code is 0 (as with the Model I) or 1, because the decrementing would produce a negative or 0 value, respectively. The negative value (-1) would trigger an "illegal function" error, and 0 would cause 255 repetitions instead of none.

Once you make it through the minefield of trouble codes, it is possible for graphics programming on the dot-matrix printer to go smoothly and even be rewarding instead of punishing. But if you've gathered from this section that it may take a little luck at first, you've gotten the message.

4

The Bit-Image Building Blocks

WITH a dot-matrix printer we can build a huge variety of graphic creations out of dots or dot patterns. These are the basic building blocks we have at our disposal in bit graphics, and they are all we need for building lines, curves, figures, symbols, alphabets, graphs—even maps and pictures.

The Binary Dot Codes

Every dot pattern has a code number called a dot code. For each time that we want the printer's printhead to make a mark on the paper, we can choose any or all of its pins to fire. By selecting the right dot code we can fire any combination of the pins we want to.

How many combinations are there? If we can control eight pins, as on the IBM, Epson, and Apple Imagewriter printers, we have a selection among 256 different patterns. Printers with seven addressable pins, such as the Tandy printers, have 128 patterns. We'll often have reasons to use only seven pins even on the eight-pin printers, so it's a good idea to learn how the dot codes are arranged on both sizes.

The IBM-Epson dot codes. Figure 4-1 displays all 256 dot patterns and their codes for printers in the IBM-Epson family. The code numbers for these printers are given just above each dot pattern in this figure, in the rows labeled EPSON CODE. Below the patterns in the upper half of the figure are three more rows of numbers, the first and third of which tell

IBM - EPSON CODES

[illegible]

FIGURE 4-1. The 256 dot patterns available in bit-graphics mode with printers in the IBM-Epson family. Codes for printers in the Apple and Tandy families and IBM-Epson double-hexadecimal (DHX\$) codes are shown underneath the patterns.

what codes the same dot patterns have on printers in the Apple and Tandy families. Between these APPLE and TRS-80 rows is still another set of codes, called DOUBLE-HEX\$ (for double hexadecimal) codes, a shorthand system that will be explained later. These three rows can be used to translate the IBM-Epson codes. In the bottom half of the table, there are only two rows under the patterns because the TRS-80 printers are limited to seven-dot patterns.

Notice several important things about the sequence of 256 dot patterns. First, they are numbered in an orderly way, from 0 (no dots) to 255 (all eight dots) in the IBM-Epson system. Although it doesn't appear so in figure 4-1, they are also numbered in an orderly way in the Apple and TRS-80 systems, but remember, those systems start numbering from the top, whereas the IBM-Epsons start at the bottom.

Notice also how the patterns change as you go across and down the chart of patterns and codes. In the first 32 patterns (top section), we have the blank code (0) and then only the bottom pin (1) firing. Moving to the right, we see that bottom pin joined by the second, third, fourth, and fifth pins from the bottom, so that by the time we reach code 31, the bottom five pins are all printing a dot. Then notice that the order of these five pins is repeated in each of the other seven sections of the chart. By the time we get to the bottom section (codes 224 to 255), the order of the bottom five pins is combined with all three top pins. When we reach code 255 (all eight dots), we've used up all the possible combinations.

This is a standard binary sequence—how a computer counts from 0 to 255 in binary code. It helps a lot to remember this when we get to using these dot patterns in graphics. The “bi” in “binary” stands for “two” (the number 2), the same meaning that “bi” has in “biplane” or “biweekly.” Counting in binary simply means counting by twos and combinations of twos squared, cubed, and raised to higher powers. What it takes to count to 128 in binary is suggested by:

$2^0 = 1$	$2^4 = 16$
$2^1 = 2$	$2^5 = 32$
$2^2 = 4$	$2^6 = 64$
$2^3 = 8$	$2^7 = 128$

It helps even more in programming to label the pins in binary fashion. Notice, to the left of the dot patterns in figure

4-1, that the top pin is labeled 128, the second pin 64, the third pin 32, and so on, down to the 1 pin at the bottom. The designers of the printer would call the fifth pin from the top "the 16 pin" instead of "pin 5." Getting into the habit of calling the pins by their binary labels ("the 64 pin," "the 4 pin," etc.) makes it much easier to work out the codes for a drawing or graph.

In ordinary arithmetic there are usually several different combinations of numbers that can add to the same total. For example, a total of 15 can be the result of adding 8, 6, and 1 or of adding 11, 2, and 2 or other combinations. In the binary counting system the rules are set up so that a total can only be reached in a single way. That way, each total stands for a unique combination of ingredients.

The first rule says that the ingredients must be whole-number powers of two (such as 2^3 , 2^6 , 2^4 , etc.). The total of 10, for example, can only be made up of the combination of 8 and 2, not by 5 and 5, or 7 and 3, because the latter are not whole-number powers of two. This means we have to make up all of the totals from 0 to 255 out of the numbers 1, 2, 4, 8, 16, 32, 64, and 128.

A second rule is that any one of these numbers can only be used once in making up any total. The sum of 22, for example, must come from $16 + 4 + 2$, not from $16 + 2 + 2 + 2$ or any other combination.

This is just the perfect arrangement to have for numbering our dot patterns. With the pins of the printhead numbered in binary fashion, any code number from 0 to 255 that is assigned to a dot pattern will tell us precisely which combination of dots will be printed by that dot code in a single print. Thus, if the dot code is 20, only the firing of the 4 pin and the 16 (third and fifth from the bottom) will occur; no other combination of the numbers 1, 2, 4, 8, 16, 32, 64, or 128 used singly will add to 20. If the code number is 241, the only combination is $1 + 16 + 32 + 64 + 128$, meaning that the bottom pin and the top four pins will print.

When you are using these code numbers in programming a graph, you usually won't be asking yourself such questions as, "Which pins print with code 223?" The question is more likely to be the other way around: "What code number do I need for printing the top and bottom pins?" (To make the sides of a bar in a bar graph, for instance.) The binary system

gives you an easy way to find the code number without even looking at a chart like figure 4-1—just add the pin numbers. For the top and bottom dots, that would be $128 + 1 = 129$.

So if you've got a lot of coding to do and you're stranded somewhere without your code chart—on a life raft in the North Atlantic, perhaps—there's nothing to worry about; just use simple arithmetic.

The Apple codes. In figure 4-2 the dot codes for the Apple-family printers are presented. Here we have 256 patterns numbered from the top instead of the bottom, but otherwise there are no important differences between figures 4-1 and 4-2. The numbering from the top makes the Apple codes exact mirror images of the IBM-Epson codes. Code 13 in figure 4-2, for example, fires the first, third, and fourth pins from the top on the Imagewriter, whereas in figure 4-1 you can see that code 13 fires the first, third, and fourth pins from the bottom on IBM-Epson printers. This makes the sequence of codes in the first row under the dot patterns the same in both figures, and thus either figure can be used to convert Apple codes to Epson codes or vice versa. A more important use of these conversions is likely to be when you are specifically interested in mirror-image coding rather than in converting programs from one printer to another.

Again, only the upper half of the Apple chart contains TRS-80 codes. But notice now that the TRS-80 codes are in order (from 128 to 255) as well as the Apple codes—both printer types are “low bit up,” that is, their dot patterns number from the top. This means that to convert any Apple dot code in the range of 0 to 127 to a TRS-80 code, just add 128.

Double-hexadecimal codes are also provided by figure 4-2—codes A0 through HV for Apple printers and A0 through DV for TRS-80 printers. Don't make the mistake of thinking that these double-hex codes print the same patterns as the double-hex codes in figure 4-1. What will happen, as a matter of fact, is that the Apple double-hex codes will print mirror images of the IBM-Epson double-hex codes.

We are indebted (again) to Francis Kalinowski (*80 Micro*, November 1983) for the double-hex shorthand system. In programs involving many hundreds of dot codes, as with his four-color drawings mentioned in chapter 1, there is too much memory consumed and too much typing to tolerate if

APPLE CODES

APPLE CODE:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1:	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
2:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
4:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
8:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
16:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
32:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
64:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
128:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
EPSON CODE:	0	128	64	192	32	160	96	224	16	144	80	208	48	176	112	240	8	136	72	200	40	168	104	232	24	152	88	216	56	184	120	248
DOUBLE-HEX:	40	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV
TRS-80 CODE:	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159

APPLE CODE:	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
2:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
4:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
8:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
16:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
32:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
64:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
128:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
EPSON CODE:	4	132	68	196	36	164	188	228	28	148	84	212	52	180	116	244	12	148	76	204	44	172	188	236	28	156	92	220	68	188	124	252
DOUBLE-HEX:	58	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	B0	BE	BF	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	B0	B1	B2
TRS-80 CODE:	168	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

APPLE CODE:	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
1:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
2:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
4:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
8:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
16:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
32:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
64:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
128:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
EPSON CODE:	2	138	66	194	34	162	98	226	18	146	82	210	50	178	114	242	10	138	74	202	42	170	186	234	26	154	90	218	58	186	122	250
DOUBLE-HEX:	08	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV
TRS-80 CODE:	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223

APPLE CODE:	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
1:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
2:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
4:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
8:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
16:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
32:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
64:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
128:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
EPSON CODE:	0	134	78	198	38	166	182	230	22	150	86	214	54	182	118	246	14	142	78	206	46	174	110	238	38	158	94	222	62	190	126	254
DOUBLE-HEX:	08	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	D0	DE	DF	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	D0	D1	D2
TRS-80 CODE:	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

APPLE CODE:	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
1:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
2:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
4:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
8:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
16:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
32:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
64:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
128:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
EPSON CODE:	1	129	65	193	33	161	97	225	17	145	81	209																				

codes are sent to the printer in the normal LPRINT CHR\$(DC) way. Kalinowski's system converts every CHR\$(DC) to a two-character string. This is quite a saving, since every CHR\$(DC) is seven, eight, or nine characters long (depending on whether DC, the dot code, is one, two, or three digits long). As you can see from the sequences in both figures, the naming of the codes is not like either the decimal or hexadecimal codes in the ASCII system. Every 32 dot patterns, the first of the two characters (always a letter) changes (from A to B to C, etc.); for the second character, the first ten are numbered 0 to 9, and the last 22 are lettered A to V within each 32-pattern group.

Converting from decimal dot codes to double-hex codes is fairly simple. After all variables beginning with A through H have been defined as string-type by a DEFSTR (A-H) statement at the beginning of the program, the two-character codes (A0, A1, A2, etc.) are assigned, in order, to the decimal dot-codes. This can be done by a series of 256 simple definitions—all the way from A0=CHR\$(0) to HV=CHR\$(255). It can also be done, less tediously, by a conversion program in BASIC, as we'll see in a later section on code conversions in general.

TRS-80 dot-codes. Although we've already had two presentations of TRS-80 codes in the IBM-Epson and Apple charts, a third chart (figure 4-3) has been prepared to focus on these codes more clearly. In this chart the 128 TRS-80 codes ranging from code 128 to code 255 are given above each dot pattern. The first two rows underneath the patterns present the corresponding codes for the upper seven of the eight dots of Apple-type printers and the lower seven of the eight dots of IBM-Epson printers. The bottom row in each section of the chart, labeled TRS-80 B-W, shows what TRS-80 code is the *black-to-white reversal* code for each of the original TRS-80 codes.

Each of the bottom three rows has an important relationship to the top row. The EPSON (1-64) row not only translates the original TRS-80 codes into IBM-Epson dot codes but also gives the mirror-image TRS-80 code if 128 is added to each of those. The mirror-image codes we had in figures 4-1 and 4-2 had to do with eight-dot codes; here we are dealing with seven-dot codes. The APPLE (1-64) row gives not only

TRS-80 CODES

TRS-80 CODE:	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
EPSON (1-64):	0	64	32	96	16	80	48	112	8	72	40	184	24	88	56	128	4	68	36	100	20	84	52	116	12	76	44	188	28	92	60	124
APPLE (1-64):	8	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
TRS-80 B-W:	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224

TRS-80 CODE:	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191								
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
EPSON (1-64):	2	66	34	98	18	82	50	114	10	74	42	186	26	90	58	122	6	70	38	102	22	86	54	118	14	78	46	182	30	94	62	126
APPLE (1-64):	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
TRS-80 B-W:	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192

TRS-80 CODE:	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
EPSON (1-64):	1	65	33	97	17	81	49	113	9	73	41	185	25	89	57	121	5	69	37	101	21	85	53	117	13	77	45	189	29	93	61	125
APPLE (1-64):	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
TRS-80 B-W:	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160

TRS-80 CODE:	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
EPSON (1-64):	3	67	35	99	19	83	51	115	11	75	43	187	27	91	59	123	7	71	39	103	23	87	55	119	15	79	47	111	31	95	63	127
APPLE (1-64):	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
TRS-80 B-W:	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128

FIGURE 4-3. The 128 dot patterns available with TRS-80 printers. IBM-Epson and Apple codes for corresponding seven-dot patterns are shown in the lower rows, along with black-white reversal (B-W) codes for TRS-80 printers.

the Apple translation but the “binary sum” of each original TRS-80 dot code.

“Binary sum” simply refers to the total you get when you add up the binary values of the printhead pins, as with the 8 pin and the 32 pin adding up to a binary sum of 40, for example. Binary sums for the TRS-80 printers are these totals before they have been increased by 128 in order to be recognized by the printer, and, as just noted, they are equivalent to the codes for the top seven pins on Apple printers. In many programming situations TRS-80 users will find it easier to use these binary-sum codes than the actual (increased by 128) TRS-80 dot codes.

Black-white reversals. For any dot pattern on any dot-matrix printer there is a “reversal pattern” in which what was black becomes white and vice versa. This is the same as saying that whatever pins fired in the original pattern did not

fire in the reversal pattern, and the pins that did not fire in the original did produce dots in the reversal.

There are 128 examples of this in figure 4-3. Notice that the B-W (reversal) code for the first dot pattern (code 128, no dots) is 255 (all seven dots). Near the middle of the chart, code 190 prints all but the top and bottom dots, whereas its reversal code (193) prints *only* the top and bottom dots.

It's more important to notice that these B-W reversal codes drop from 255 to 128 in a regular order while the original TRS-80 codes increase from 128 to 255 and the binary sums (Apple codes) increase from 0 to 127. The formula for converting the original dot codes (DC) to their black-white reversal codes (BW), then, is simple: $BW = 383 - DC$. And to get to the BW numbers from the binary sums (BS), just subtract from 255 instead of 383: $BW = 255 - BS$.

When the IBM-Epson and Apple printers are being used as seven-dot printers (i.e., when their 128 pins are not used), converting their dot codes to black-white reversal codes is just as easy: subtract the dot code from 127. When all eight pins on these printers are being used, in which case their dot codes range from 0 to 255 instead of 0 to 127, subtract from 255 to get the black-white "opposite numbers."

By the way, black-white reversal codes shouldn't be confused with mirror-image codes. (And they might be, with seven- or eight-dot printers, because some of these are identical.) Substituting dots for blanks and blanks for dots, which is what black-white reversal amounts to, is quite different from merely turning a dot pattern upside down.

If we program a bunch of dot codes to print a simple figure in black on white paper, can we also turn that drawing into a white figure on a black background just by subtracting the dot codes from a constant? You know what the answer is going to be, but let's see how it works.

PROGRAM 4-1. ARROW. Printing a simple bit-image drawing.

```

10 'PROGRAM 4-1: "ARROW" -- TO DEMONSTRATE PRINTING A SIMPLE BIT-IMAGE
15 '                                     FORM ON THE EPSON FX-80 PRINTER
20 LPRINT CHR$(27)"@";                  'MASTER RESET
30 LPRINT CHR$(27)"K"CHR$(30)CHR$(0);   'RESERVE 30 COLUMNS FOR GRAPHICS
40 READ DC: IF DC<0 THEN END            'READ DOT-CODE
50 LPRINT CHR$(DC);                     'PRINT DOT-PATTERN
60 GOTO 40                               'READ AND PRINT ANOTHER
70 DATA 0,0,34,54,62,62,62,62,62,62,28,8,8,8,8
80 DATA 8,8,8,8,8,8,8,42,62,28,28,8,8,0,0,-1

```

Program 4-1, ARROW, uses the bottom seven pins of an IBM-Epson printer to draw the figure in the normal black-on-white fashion. With an Apple printer line 20 would have to be `LPRINT CHR$(27)"G0030"`; and in Applesoft BASIC all three LPRINTs (lines 20, 30, and 50) would have to be PRINTs after PR#1. TRS-80 users should change line 20 to `LPRINT CHR$(18)` and line 40 to `LPRINT CHR$(DC+128);`.

If we run this program, our printout should appear like figure 4.4: An arrow it is. Not a very big one for all those dot codes (Apple Imagewriter users, if they use the 160-per-inch dot density, might want a magnifying glass to see the arrow in all its glory), but don't let that upset you—we'll see some shortcuts for bit-image drawing in the next chapter.



FIGURE 4-4. A simple bit-image figure.

Now if we just change what's inside the parentheses of `CHR$(DC)` in line 50 to `127-DC` for the IBM-Epson and Apple printers and to `255-DC` for a TRS-80 printer, we should see what's shown in figure 4-5.



FIGURE 4-5. Black-white reversal of figure 4-4's arrow.

And there's the flip-flop—easier to do than a photographic reversal. Notice that if we hadn't had those two 0 codes at each end of the data, we wouldn't have had the ends of our white-on-black arrow showing up against a dark background.

This is enough to demonstrate the principle and method of black-white reversal. Considering how often such reversals are done by photography in commercial graphics, this demonstration was worth pursuing, especially because there's no reason why much larger printouts by the computer can't be reversed the same simple way.

Most often used dot codes. Another way of getting acquainted with the graphic dot codes is to see what they look like when they are repeat-printed as well as printed only once. Figure 4-6 displays all 128 of them on the TRS-80 DMP-400. For each code, this figure shows a single print,

CODE #	BS	CODE #	BS	CODE #	BS	CODE #	BS
128	0	160	32	192	64	224	96
129	1	161	33	193	65	225	97
130	2	162	34	194	66	226	98
131	3	163	35	195	67	227	99
132	4	164	36	196	68	228	100
133	5	165	37	197	69	229	101
134	6	166	38	198	70	230	102
135	7	167	39	199	71	231	103
136	8	168	40	200	72	232	104
137	9	169	41	201	73	233	105
138	10	170	42	202	74	234	106
139	11	171	43	203	75	235	107
140	12	172	44	204	76	236	108
141	13	173	45	205	77	237	109
142	14	174	46	206	78	238	110
143	15	175	47	207	79	239	111
144	16	176	48	208	80	240	112
145	17	177	49	209	81	241	113
146	18	178	50	210	82	242	114
147	19	179	51	211	83	243	115
148	20	180	52	212	84	244	116
149	21	181	53	213	85	245	117
150	22	182	54	214	86	246	118
151	23	183	55	215	87	247	119
152	24	184	56	216	88	248	120
153	25	185	57	217	89	249	121
154	26	186	58	218	90	250	122
155	27	187	59	219	91	251	123
156	28	188	60	220	92	252	124
157	29	189	61	221	93	253	125
158	30	190	62	222	94	254	126
159	31	191	63	223	95	255	127

FIGURE 4-6. Printouts of the 128 TRS-80 seven-dot patterns, each printed once and fifty times.

resulting from `LPRINT CHR$(DC)`, and how the code looks when repeated fifty times by the command `LPRINT STRING$(50,DC)`.

After studying these, you won't be surprised to learn that some of the codes are hardly ever used in graphics, and others are used over and over. Codes that print just a single dot are often used to draw simple horizontal lines in graphs or diagrams, for example, and bar graphs will make heavy use of the all-dots code, 255. (Notice, incidentally, that despite all the differences in codes among the IBM-Epson, Apple, and Tandy printers, code 255 is the same for all three families in printing all dots that are addressable.) We'll seldom have much use for the code that looks like an exclamation point (TRS-80 code 223, Epson code 125, Apple code 95), but its next-door neighbor (224, 3, and 96, respectively) is a big favorite for drawing the vertical axis of a graph.

The codes that are most likely to be needed in graphics are separated from the others in figure 4-7. It's very handy to

Conversions of dot codes. Sooner or later you're going to need more efficient ways of converting from IBM-Epson to Apple or TRS-80 dot codes than looking up numbers in charts like figures 4-1 to 4-3. We've already noted that the first half of the Apple codes can easily be converted to TRS-80 codes by just adding 128 and that you can get those Apple codes by subtracting 128 from the TRS-80 codes. There is no simple formula, however, for converting IBM-Epson codes to their Apple or Tandy equivalents.

But there are BASIC programs for doing these conversions. This type of conversion, remember, is the same as converting codes to their mirror-image codes, so the programs can serve that purpose as well. Program 4-2, EPSTOAPL ("Epson to Apple"), converts all the eight-dot codes from the IBM-Epson to the Apple numbering system, or vice versa. Program 4-3, TRSTOEPS, converts the seven-dot binary sums for the TRS-80 or Apple printers to the first half of the IBM-Epson codes (0 through 127).

PROGRAM 4-2. EPSTOAPL. Converting IBM-Epson dot codes to Apple dot codes.

```

5000 'PROGRAM 4-2: "EPSTOAPL",A -- CONVERSION OF EPSON TO APPLE DOT-CODES
5005 'Copyright (c) 1985 by John Warner Davenport
5010 'SUBROUTINE FOR IBM PC, APPLE, OR TRS-80 COMPUTERS
5012 'REMOVE IN-LINE REMARKS AFTER APOSTROPHES IF APPLESOFT BASIC IS USED
5020 NEPS=A(I): PRINT "ORIGINAL CODE =" ;NEPS      'ENTER EPSON DOT-CODE, A(I)
5030 FOR P=7 TO 0 STEP -1                          '(8 PINS, NUMBERED 7 TO 0)
5040   X(P)=NEPS-2^P                                '(CONVERSION OF DECIMAL-
5050   IF X(P)<0 THEN Y$(P)="0": GOTO 5070          'MAL CODE TO
5060   Y$(P)="1": NEPS=X(P)                          'BINARY FORM)
5070 NEXT P
5080 BYTE$=Y$(7)+Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0)  'ORIGINAL
5090 PRINT "ORIGINAL BYTE =" ;BYTE$                  'DISPLAY BINARY BYTE
5100 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6)+Y$(7) 'MIRROR IMAGE
5110 PRINT "MIRROR-IMAGE BYTE =" ;MBYTE$             'DISPLAY MIRROR-IMAGE BYTE
5120 NAPL=128*VAL(Y$(0))+64*VAL(Y$(1))+32*VAL(Y$(2))+16*VAL(Y$(3))+8*VAL(Y$(4))+
    4*VAL(Y$(5))+2*VAL(Y$(6))+VAL(Y$(7))              'CONVERT MIRROR-BYTE TO DECIMAL
5130 B(I)=NAPL                                         'TAG CONVERTED CODE AS B(I)
5140 PRINT "ORIGINAL CODE, ";A(I);", EQUALS APPLE CODE ";B(I)
5150 RETURN

```

PROGRAM 4-3. TRSTOEPS. Converting TRS-80 dot codes to IBM-Epson dot codes.

```

5000 'PROGRAM 4-3: "TRSTOEPS",A -- MIRROR-IMAGE CONVERSION OF 7-BIT DOT-CODES
5005 'Copyright (c) 1985 by John Warner Davenport
5010 'SUBROUTINE FOR IBM PC OR TRS-80 MODEL 4 (USES '^' SYMBOL)

```

```

5020 NTRS=N: PRINT "ORIGINAL DECIMAL:";NTRS      'ENTER TRS-80 CODE, N
5030 FOR P=6 TO 0 STEP -1                        '(7 PINS, NUMBERED 6 TO 0)
5040   X(P)=NTRS-2^P                             '(CONVERSION OF
5050   IF X(P)<0 THEN Y$(P)="0": GOTO 5070        ' DECIMAL CODE
5060   Y$(P)="1": NTRS=X(P)                      ' TO BINARY FORM)
5070 NEXT P
5080 BYTE$=Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0) 'ORIGINAL BYTE (BINARY)
5090 PRINT "ORIGINAL BYTE$:";BYTE$
5100 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6) 'TURN BYTE UPSIDE DOWN
5110 PRINT "MIRROR-IMAGE BYTE$:"MBYTE$
5120 EC=64*VAL(Y$(0))+32*VAL(Y$(1))+16*VAL(Y$(2))+8*VAL(Y$(3))+4*VAL(Y$(4))+2*VA
    L(Y$(5))+VAL(Y$(6))                          'CONVERT MIRROR BYTE TO DECIMAL
5130 PRINT "ORIGINAL TRS CODE, ";NTRS;"", EQUALS EPSON CODE ";EC
5140 RETURN

```

The essence of these programs is the mirror-image principle combined with expanding decimal codes to their binary forms. That is, in each program the dot code to be converted is first changed from its decimal form to a set of 0 and 1 bits representing its binary form (lines 5030–5080 in both programs). For program 4-2 the binary form is eight-bit and for program 4-3 it is seven-bit. In both cases, once the the sequence of 0's and 1's represening a dot code has been calculated and stored in an array, it is turned upside down by reversing the subscript of the array (line 5100), so that the 0 or 1 status of the top bit becomes the status of the bottom bit, the second bit's status becomes that of the next-to-last bit, and so forth. After the reversal, there is a final conversion of the binary form to the decimal form, to determine the new identity of the dot code.

Using program 4-2, if we want to convert Epson code 207 to the Apple code that prints the same eight-dot pattern (the top two and bottom four pins printing dots, the third and fourth pins from the top not printing—see figure 4-1), the program will calculate the binary form of code 207, which is 11001111. Then the subscript reversal will turn that into its mirror-image, 11110011. Finally, this new binary code will be converted to its decimal value, 243, by the formula in line 5120. Code 243 is the Apple code that will print the pattern that has two dots at the top, four at the bottom, and two blanks in between (you can check this in figure 4-2). It is also the case that decimal code 243 is the Epson code that prints the mirror-image of the pattern printed by Epson code 207 (check figure 4-1).

Converting seven-dot codes, which program 4-3 does, in-

volves all the same steps as program 4-2, the only difference being one less bit in the binary forms. Notice that if you are starting with a TRS-80 code, which would have to be a binary sum in the range of 0 to 127, you come out with a seven-bit Epson code that is also in the 0-to-127 range, and thus it is the bottom seven pins on the Epson that you are dealing with.

As promised, we also have a BASIC program for converting decimal dot codes to Kalinowski's double-hexadecimal codes. This is program 4-4, DBLHEXER.

PROGRAM 4-4. DBLHEXER. Converting dot codes 0-255 to double-hexadecimal abbreviations.

```

14000 'PROGRAM 4-4: "DBLHEXER",A -- CONVERSION OF DECIMAL TO DOUBLE-HEX CODES
14002 'Copyright 1985 by John Warner Davenport
14005 'SUBROUTINE FOR IBM, APPLE, OR TRS-80 COMPUTERS
14008 'REMOVE IN-LINE REMARKS AFTER APOSTROPHES IF APPLESOFT BASIC IS USED
14010 DEFSTR A-H
14020 AC%=FIX(DC%/32)+65: L%=CHR$(AC%)      '(AC%=ASCII CODE, DC%=DECIMAL CODE)
14030 FOR Z=0 TO 9                          '(L%=LEFT HALF OF DOUBLE-HEX CODE)
14032   FOR X=0 TO 224 STEP 32
14035     IF DC%=Z+X THEN R%=STR$(Z): GOTO 14070
14038   NEXT X
14040 NEXT Z: NEXT X                        '(R%=RIGHT HALF OF DOUBLE-HEX CODE)
14050 FOR SC=65 TO 86
14052   FOR X=-55 TO 169 STEP 32
14055     IF DC%=SC+X THEN RZ=CHR$(SC): GOTO 14070
14058   NEXT X
14060 NEXT SC: NEXT X
14070 DX=L%+RIGHT$(R%,1)
14080 IF DC%=183 THEN DX="FN"               'CHANGE RESERVED WORD 'FN' TO 'FW'
14090 PRINT "ORIGINAL CODE";DC%; " = DHEX CODE ";DX
14100 RETURN

```

This program, which would normally be used as a subroutine in a larger program, is for converting the 256 IBM-Epson or Apple dot codes. It makes use of the ASCII table (see table 2-3), specifically ASCII decimal codes 48 to 57 for numerals 0 to 9 and codes 65 to 86 for capital letters A to V, skipping the intervening codes (58 to 64), which stand for punctuation marks. Since letters A through H are used for the first character in the two-character code, the converting statements in the program are preceded by **DEFSTR A-H** so that all 256 hexadecimal codes can be referred to as strings without the type-declaration \$ as a third character.

DBLHEXER then goes through an AC-DC routine to translate the first of the two characters. Actually it's the other way around (DC-to-AC), and % signs have to be added to these so that the computer will treat them as numerical variables. Each DC% (dot code or decimal code) is converted to an ASCII code (AC%) by the formula $AC\% = \text{FIX}(DC\%/32) + 65$ in line 14020, and then the AC% values are in turn converted to letters A through H by the BASIC's CHR\$() function.

One of those letters becomes the first (left side) of the two characters in a double-hex code by the definition L\$ = CHR\$(AC%) in line 14020. For the second (right) character, called R\$, the program uses the STR\$ function for digits 0 through 9 and the CHR\$ function for letters A through V. Finally, the double-hex code (DX) is assembled by concatenating the "left" (L\$) and "right" (R\$) halves, that is, by $DX = L\$ + \text{RIGHT}\$(R\$, 1)$ in line 14070. One DX code has to be changed—code 183—to avoid using one of BASIC's reserved words, FN (changed to FW).

A program that merely translates an Epson or Apple code into a double-hexadecimal code doesn't go far enough to let you print different dot patterns by sending different double-hex codes to the printer. In order to get these two-character codes to do what you want them to, the appropriate CHR\$(DC) values have to be assigned to them, so that the computer will "know" that A2 means CHR\$(2), GD means CHR\$(205), and so forth. As we noted earlier, these assignments can be made by a series of 256 definitions typed into the program, of the form $DX = \text{CHR}\$(DC)$.

PROGRAM 4-5. DHEXPRNT. Bit-image printing using double-hexadecimal codes.

```

10 'PROGRAM 4-5: "DHEXPRNT" -- ASSIGNMENT & PRINTING OF DOUBLE-HEX CODES
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR EPSON PRINTERS DRIVEN BY IBM PC (USE PRINT# INSTEAD OF LPRINT, INITIALI
    E WITH 'OPEN "LPT1:" AS #1: WIDTH #1, 255) OR TRS-80 (USE BOOTHE'S DRIVER)
20 GOSUB 15000
30 GOSUB 15170
40 STOP
15000 '***** ASSIGNMENT OF CHR$(0-255) TO 256 DOUBLE-HEX CODES *****
15010 DEFSTR A-H
15020 DIM Z$(11),DX(256),DY(256)
15030 I=0
15040 FOR LC=65 TO 72: Q=0
15050   FOR RC=48 TO 86
15060     IF RC>57 AND RC<65 THEN Q=Q+1: GOTO 15140

```

	'ASCII CODES FROM A TO H
	'ASCII CODES FROM 0 TO V
	'SKIP 58-64 (PUNCTUATION)

```

15070  L$=CHR$(LC): R$=CHR$(RC)                    'LEFT & RIGHT BYTES
15080  IF L$="F" AND RIGHT$(R$,1)="N" THEN R$="W"   'CHANGE 'FN' TO 'FW'
15090  DX(I)=L$+RIGHT$(R$,1)                    'STORE BYTE IN DX(I) ARRAY
15100  J=32*FIX(I/32)                    '(J AND Q ARE COUNTERS)
15110  DY(I)=CHR$(RC-48-Q+J)                    'STORE CHR$( ) IN DY(I) ARRAY
15120  PRINT "DOUBLE-HEX CODE, DX(I), = ";DX(I);"   ASC(DY(I)) = ";ASC(DY(I))
15130  I=I+1
15140  NEXT RC
15150  NEXT LC
15160  RETURN
15170  '***** TEST PROGRAM: BIT-IMAGE PRINTING FROM DOUBLE-HEX DATA *****
15180  FOR N=1 TO 11
15190  READ Z$(N): PRINT Z$(N)                    'READ AND DISPLAY DOUBLE-HEX CODE
15200  IF Z$(N)="END" THEN GOTO 15260
15210  FOR I=0 TO 255
15220  IF Z$(N)=DX(I) THEN Z$(N)=DY(I)           'CONVERT DHEX TO CHR$( ) STRING
15230  NEXT I
15240  NEXT N
15250  DATA AS,AK,B2,B2,B0,B0,B0,B2,B2,AK,AS,END
15260  LPRINT CHR$(27)"@";CHR$(27)"L"CHR$(11)CHR$(0);   'RESET, RESERVE 11 COLS.
15270  FOR N=1 TO 11: LPRINT Z$(N);: NEXT N           'PRINTS LETTER "C"
15280  LPRINT CHR$(13);CHR$(10);
15290  FOR N=1 TO 11: LPRINT ASC(Z$(N));: NEXT N           'PRINTS DECIMAL CODES
15300  LPRINT CHR$(13);CHR$(10);
15310  RETURN

```

But you'll have a whole lot less typing if you use program 4-5 instead. This program, called DHEXPRNT, can be used with or without the DBLHEXER program. It uses two string arrays, DX(I) and DY(I); two ASCII-code variables, LC and RC (for "left character" and "right character"); and three counter variables, I, J, and Q. With nested FOR-NEXT loops the program goes through all 256 possible combinations of left characters (ranging from A to H) with right characters (0 to 9, A to V) that can make up a double-hex code, DX(I). While doing that it assigns a decimal code from 0 to 255 to the variable DY(I). The arithmetic of assigning these decimal codes is handled by the J and Q counters and the statement `DY(I)=CHR$(RC-48-Q+J)` in line 15110.

DX(I) and DY(I) become linked by the subscript (I) that they have in common. If the value of I is, say, 87, DX(87) will be equal to CN and DY(87) will equal CHR\$(87). Setting DX(87) equal to DY(87) will not make CN equal to CHR\$(87)—it will just make DX(87) equal to DY(87)—but if you bring in a third string variable called Z\$(N), you can get the printer to print the dot pattern of CHR\$(87) whenever DX(87) turns up.

The test program in subroutine 15170 of DHEXPRNT demonstrates this. It starts up another FOR-NEXT loop for setting Z(N)$ equal to successive DATA values, which are in double-hex form. The loop runs from $I=0$ to 255 and contains the statement **IF Z(N)=DX(I)$ THEN Z(N)=DY(I)$** in line 15220. Any DATA values of Z(N)$ in the range of A0 to HV can thus be printed out as dot patterns. For example, in lines 15250–15270 the sequence of DATA values AS, AK, B2, B2, B0, B0, B0, B2, B2, AK, AS operated on by **LPRINT Z(N)$** produces a printout of the letter C, and reading the same sequence to the command **LPRINT ASC(Z(N)$)** in line 15290 produces a printout of the dot codes: 28 20 34 34 32 32 32 34 34 20 28.

While programs 4-4 and 4-5 can be used for converting and assigning Apple codes as well as IBM-Epson codes, remember that the dot patterns will be different for the two types of printer, except for a few that happen to be vertically symmetrical, i.e., for which the mirror-image is the same as its original image.

TRS-80 codes can also be translated and assigned to double-hex codes with programs similar to these. The main changes would be subtracting 128 from each TRS-80 code to get values of DC% in program 4-4 and adding 128 to the value of $RC-48-Q+J$ in line 15110 of program 4-5. That same line could be altered for black-white reversals by subtracting the value of $RC-48-Q+J$ from 255 on the IBM-Epson and Apple printers and the value of $RC-48-Q+J+128$ from 383 on the TRS-80 printers.

Kalinowski goes further than using double-hex codes as single dot codes. He defines what you might call higher-order double-hex codes in terms of combinations of double-hex codes. After using a **DEFSTR A-Z** statement and assigning the codes A0 through HV to decimal codes 0 through 255, he defines Z1 as single full column of dots—which can be done by **Z1=HV** or **Z1=CHR\$(255)**. Then he goes on to define two consecutive eight-dot prints by **Z2=Z1+Z1** (concatenation), three consecutive by **Z3=Z2+Z1**, and so forth, working up to ten consecutive (**ZA=Z5+Z5** for $10=5+5$). Then tens, twenties, thirties, etc., are combined to get up to 100 (**ZJ=ZI+ZA** for $100=90+10$).

With this scheme we can order the printer to print, say, 108 dot columns filled with dots by **LPRINT ZJ+Z8** instead of

having to type Z1 (or HV) 108 times or using `LPRINT STRING$(108,255)`. Kalinowski has a similar scheme for blank spaces, starting with `Q1=A0` or `Q1=CHR$(0)`, and for other repeat codes. He also converts all the Epson control codes he needs to double-hex codes; instead of using `LPRINT CHR$(27)"K"` for entering graphic mode with 60-dots-per-inch density, for example, he can use the command `LPRINT BY`, after `BY` has been defined by `BY=AR+CB` or `BY=CHR$(27)+"K"` or `BY=CHR(27)+CHR$(75)`.

After all conversions and higher-order definitions have been made, the double-hex coding system provides a better-than-80% saving in number of characters to be typed or held in memory compared to using decimals in `LPRINT CHR$()` commands. (But since there are other shortcuts that provide similar savings and keep programs easier to follow as well, we will be very sparing in our use of the double-hex system in this book.)

Bit-Image Screen Dumps

Before getting totally immersed in bit-graphics programming, we should take some time to enjoy the dot-addressable printer when it's used "the easy way"—dumping the contents of the computer's screen to the printer. Actually, it's not necessarily easier to program graphics from scratch on the screen than directly on the printer, but when someone else does most of the screen programming for you, it figures to be a breeze. And yet even with that help it probably won't be, because the screen-dumping procedure itself has to be programmed or purchased.

Several commercial screen-dump packages for graphics, varying with the system, have been on the market for quite a while. Assembly-language programs for screen-printing graphic images have also appeared in the microcomputer magazines at regular intervals. But the do-it-yourselfer who is penniless and unfamiliar with assembly language has reasons to be interested in BASIC programming of screen dumps.

This will require some tolerance of long waiting periods, because BASIC is probably the slowest language to use. But if the resolution of the screen is low, there won't be so many screen units to dump to the printers that it will take more

than a few minutes. Unfortunately, the lower the resolution the less the screen graphic will be worth dumping, but if you shift to a high-resolution computer it's likely to take hours instead of minutes.

Let's start with low resolution. Each dot of light, called a pixel (for "picture element"), on the computer's screen needs to be turned into a dot of ink on the printer's page. How big a pixel can be, in addressable form, varies tremendously with the computer, and we are at the mercy of that computer's design. The size and even the shape of the printer's dot are much more under our control, as we'll soon see. In the case of the TRS-80 Model III, there are 48 rows of addressable pixels, and each row has 128 pixels. The full screen therefore has 48 times 128, or 6,144 pixels in all. If that seems like a lot to you, compare it to the standard IBM PC's total of 128,000 (200 by 640) or the Tandy Model 2000's 256,000 (400 by 640).

Mike Keller (in *80 Micro*, June 1983) has presented some simple BASIC programs for representing each pixel on the Model III (or Model I) screen by a single dot on the Epson (MX-80) and Apple-type (NEC 8023A) printers. Program 4-6 is a version of his program for TRS-80 printers.

PROGRAM 4-6. MINIPRNT. Printing miniature screen dumps with the Model III.

```

5000 'PROGRAM 4-6: "MINIPRNT" SUBROUTINE -- FOR MINIATURE SCREEN DUMP
5005 'FOR TRS-80 COMPUTERS AND PRINTERS
5010 'Based on Mike Keller's "Mini-Screen" programs (80 Micro, June 1983)
5020 CLEAR 1000: DEFINT A-Z: DIM A(128,9)
5030 'FUNCTION TO CONVERT 6 VERTICAL PIXELS TO A DECIMAL (RANGE OF 0-127)
5040 DEF FN F1(X)=A(COL,0)+A(COL,1)*2+A(COL,2)*4+A(COL,3)*8+A(COL,4)*16+A(COL,5)
    *32
5050 '=====PRINT SCREEN GRAPHICS=====
5060 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18); 'CONDENSED PRINTING
5070 FOR REF=0 TO 42 STEP 6
5080   FOR COL=0 TO 127
5090     FOR ROW=0 TO 5
5100       A(COL,ROW)=-(POINT(COL,ROW+REF)) 'STORAGE OF 1 OR 0 IN 2-D ARRAY
5110     NEXT ROW
5120   NEXT COL
5130   FOR COL=0 TO 127
5140     LPRINT CHR$(18);CHR$(FN F1(X)+128); 'PRINT DOT-PATTERN
5150   NEXT COL
5160   FOR Z=1 TO 6: LPRINT CHR$(27);CHR$(50);: NEXT Z '6/72" LINE FEED
5170   LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(0); 'CARRIAGE RETURN W/O LINE FEED
5180 NEXT REF
5190 RETURN

```


These programs depend crucially on the POINT(X,Y) function, which tells you whether and which pixels are on (lighted) or off. Whenever you question the computer about a pixel's on-off status by specifying its X (horizontal) and Y (vertical) coordinates, a value of -1 is returned if it is lighted and a value of 0 is returned if it's off. This applies to a graphic-block pixel, not the smaller points of light that make up alphanumeric characters on the Model III. That means that only graphics, not text characters, will be dumped to the printer. After we have drawn a graphic figure on the screen, it can be described exactly by the X-Y coordinates of the pixels that are on.

In Keller's programs for the MX-80 and NEC 8023A, these coordinates are read eight at a time vertically, so that in effect there are only six rows by 128 columns, or 768 eight-bit light patterns to be translated into the eight-bit dot codes of those printers. Keller's neat trick is to come up with the defined function (DEF FN, in BASIC) that does the translating. You'll find a modified version of that DEF FN in line 5040 of program 4-6.

The only important change is to divide the forty-eight original rows of the Model III's screen by six instead of by eight, so that the size of the vertical pixel pattern to be translated will get down to where it can be handled by a TRS-80 printer, which is limited to seven-bit dot codes. The defined function converts the vertical pattern of 0s and 1s, which is produced by reading POINT values in each six-bit pixel column, to a decimal code in the range of 0 to 127 by successively multiplying the 0's and 1's by powers of two. The 0 or 1 value of the top pixel is multiplied by 1, the next by 2, the next by 4, and so on down to the bottom one of the six (multiplied by 32). The resulting decimal code is then sent to the printer by `LPRINT CHR$(18);CHR$(FN F1(X)+128)`.

It may come as a shock that the printed figure ends up less than an inch high, but remember that eight butted lines of six-dot-high dot patterns amount to only $\frac{48}{72}$ inch. Figure 4-8 shows some examples of these miniature screen dumps. Each of these took a little more than three minutes to produce. As Keller points out, black-white reversals can be generated by changing line 5100 to:

```
A(COL,ROW)=POINT(COL,ROW+REF)+1
```

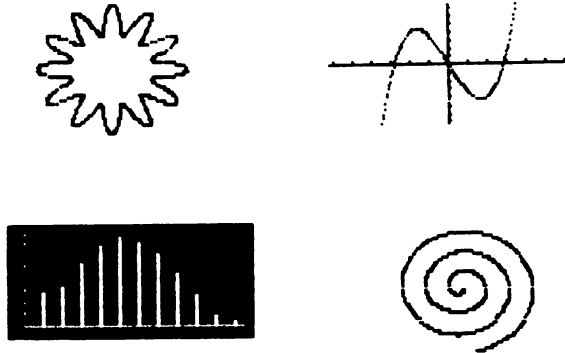


FIGURE 4-8. Miniature screen dumps produced on the DMP-400 by variations of Keller's MINIPRNT program (program 4-6).

If you want a larger printout from the same screen figure, you can get it by changing the function in DEF FN so that only two screen lines at a time are converted instead of six, and then use the printer's built-in block-graphic characters. Lines in the Model III program would have to be changed or added as follows:

```

5030 'FUNCTION TO CONVERT 2 VERTICAL PIXELS
      TO 0, 7, 8, OR 15
5040 DEF FN FL(X)=A(COL,0)*7+A(COL,1)*8
5135 IF FN FL(X)=0 THEN LPRINT
      CHR$(18);STRING$(3,128);CHR$(30);: GOTO
      5150
5140 LPRINT CHR$(30);CHR$(FN FL(X)+224);
5145 LPRINT CHR$(8);CHR$(6); 'BACKSPACE 3
      DOT SPACES

```

Adding the values 0, 7, 8, or 15 (see line 5030) to 224 (see line 5140) determines whether the printer receives codes 224, 241, 242, or 253, which are block-graphic codes for, respectively, a blank, the upper pixel, the lower pixel, and both pixels, in each two-pixel column. Instead of using the code 224 for blanks, line 5135 intervenes with STRING\$(3,128) in a temporary switch to graphic mode. When the other three codes are printed (by line 5140) they are immediately followed by a three-dot backspace, so that the left-to-right distance traveled by the printhead will be the same for prints of blocks as for blanks. This speeds up the printing as well as saving wear and tear on the printer.

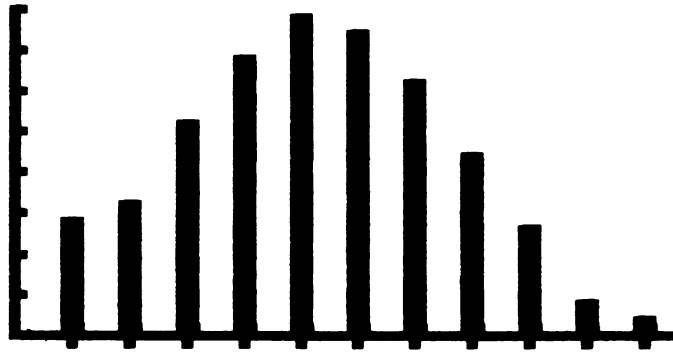
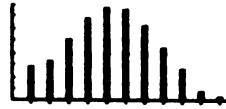


FIGURE 4-9. Increased size of a MINIPRNT type of screen dump when screen pixels are represented by block-graphic characters in text mode instead of by single dots.

The increase in size is shown by figure 4-9. Similar program changes with other printers would have to take into account the size of their built-in block-graphic characters, if they have any, and the size of line feeds for proper butting of the lines in print.

Using the text-mode graphic blocks is only a partial step toward varying the size of a screen dump. In graphic mode, the block of print that is used to represent each pixel on the screen can be as small as a single dot (as in the MINIPRNT programs) or as high as the string of dots printed by `CHR$(255)`. (That would be eight dots on the IBM-Epson and Apple printers, seven on Tandy printers.) Figure 4-10 shows the range of selection for eight-dot printers.

You can generate these block prints by sending the code for a single dot once, a two-dot code twice—e.g., `LPRINT STRING$(2,3)`—a three-dot code three times—`LPRINT STRING$(3,7)`—and so forth. Then, instead of scanning the



FIGURE 4-10. Bit-image squares of eight different sizes that can be produced by eight-dot printers.

screen six or two lines at a time, have your screen-dump program read only a single pixel line. That, at the cost of more reading and printing time, eliminates the need for any conversion of 0's and 1's to dot codes or block codes. Program 4-7, DUMPSIZE, varies the size of the bit-image block representing each pixel from one to seven dots for a TRS-80 system.

PROGRAM 4-7. DUMPSIZE. Varying the size of screen dumps with the Model III.

```

5000 'PROGRAM 4-7: "DUMPSIZE",A -- VARYING THE SIZE OF SCREEN DUMPS
5010 'SUBROUTINE FOR TRS-80 MODEL III AND TRS-80 PRINTERS
5020 DEFINT A-Z: DIM A(128,48)
5030 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18); 'CONDENSED PRINTING
5040 FOR ROW=0 TO 47
5050   FOR COL=0 TO 127
5060     A(COL,ROW)=- (POINT(COL,ROW)) 'STORES 1 OR 0 IN 2-D ARRAY
5070   NEXT COL
5080   LPRINT CHR$(30);ROW;
5090 NEXT ROW
5100 FOR SZ=2 TO 7
5110   LPRINT STRING$(3,13); 'THREE LINE FEEDS
5120   FOR ROW=0 TO 47
5130     FOR COL=0 TO 127
5140       IF A(COL,ROW)=0 THEN LPRINT CHR$(18);STRING$(SZ,128);: GOTO 5170
5150       ON SZ GOSUB 5230,5240,5250,5260,5270,5270
5160       IF A(COL,ROW)>0 THEN LPRINT CHR$(18);STRING$(SZ,DC);
5170     NEXT COL
5180   FOR Z=1 TO SZ: LPRINT CHR$(27);CHR$(50);: NEXT Z
5190   LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(0);
5200 NEXT ROW
5210 NEXT SZ
5220 RETURN
5230 DC=131: RETURN
5240 DC=135: RETURN
5250 DC=143: RETURN
5260 DC=159: RETURN
5270 DC=191: RETURN
5280 DC=255: RETURN

```

Figure 4-11 displays the different sizes of printouts produced by DUMPSIZE. The largest of these, in actual printed size, is just about the same size as the figure on the Model III's screen. Regardless of size, the printing time for each of the seven printouts is around five minutes, as opposed to MINIPRNT's taking approximately three minutes.

Unless you consider five minutes to be too long, this type of screen dump also works well with small-screen portable computers that have the POINT function, such as the EPSON HX-20. (For portables that do not, such as the Radio Shack Model 100, other ways of reading the screen—implementing the PEEK command for screen addresses in RAM, for example—will have to be used.) Using a bit-image block produced by STRING\$(3,135) in elite pitch on the DMP-400, STRING\$(5,7) in double density on the FX-80, or STRING\$(7,7) with the Imagewriter's highest density, and a $\frac{3}{72}$ -inch line feed, you'll get screen prints that are slightly larger than the HX-20's LCD display.

If you want to magnify the screen image, use a larger block (and equal-sized blank) and line feed. With seven-dot-high blocks and $\frac{7}{72}$ -inch line feeds, the HX-20's $1\frac{1}{8}$ inch \times $3\frac{1}{2}$ inch screen blows up to about 3 inch \times 11 inch (too big to fit on an eight-inch printer). You may be surprised how some of your miniature creations on the HX-20's 32-by-120-pixel screen appear in a size like that, and how much better they look in high-contrast printouts than on the rather foggy LCD screens of notebook computers.

With computers having much higher-resolution screens than the Model III or HX-20, BASIC programs depending on the POINT function in row-by-row readings will be too slow for most people. Reasonably fast dumps for the IBM PC and for Apple computers are available in commercial software for prices under fifty dollars. The Tandy Model 2000 has a subroutine on its system disk for dumping images on its 400-by-640-pixel color monitor to the Radio Shack CGP-220 color ink-jet printer.

IBM users, however, should be aware of the excellent programs written by Baenziger (*PC World*, December 1983). His GRUMP (for "graphic dump") programs combine advanced BASIC and machine language for screen prints of high-resolution graphic images. What is more, he has prepared four different versions of GRUMP for four different

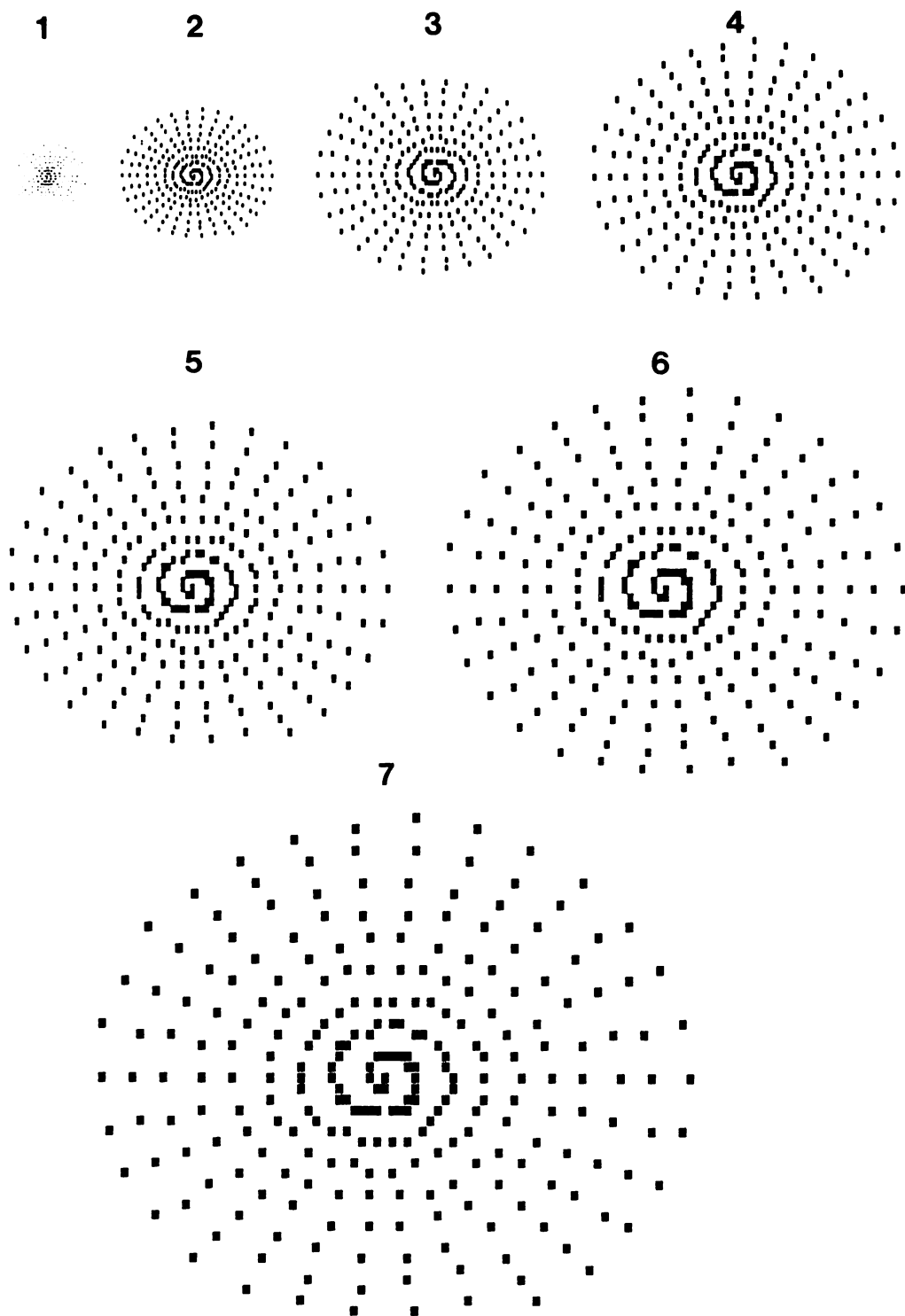


FIGURE 4-11. Different-sized screen dumps generated by the DUMPSIZE program (program 4-7).

categories of dot-addressable printers, including those with six-, seven-, and eight-pin printheads and those with low-bit-up as well as high-bit-up pin configurations. Furthermore, by using the GET statement in the PC's advanced BASIC, the programs are able to read and store about 200 columns of graphics from the screen every five seconds. So even with 200 times 640 pixels to cope with, the whole screen-dump process takes only two to three minutes.

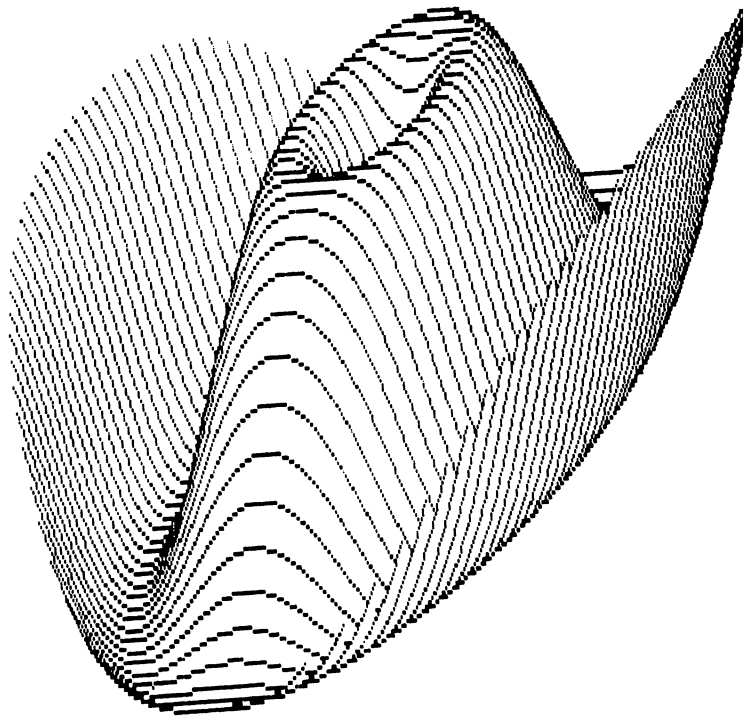


FIGURE 4-12. A bit-image screen dump of a three-dimensional hidden-line figure drawn on the IBM PC screen and printed by an Epson FX-80 printer. (Reproduced with permission of Prentice-Hall, Inc., from *Microcomputer Graphics: Techniques and Applications* by Donald Hearn and M. Pauline Baker).

As with our earlier screen-dump programs, any version of GRUMP can be tacked onto a program for screen graphics, and microcomputer books offer a tempting cafeteria of selections. Figure 4-12 shows one of the classics from Hearn and Baker's *Microcomputer Graphics: Techniques and*

Applications—a sine function in three dimensions with hidden-line removal (or you could call it a hat). This example provides a standard to be compared with printouts produced by direct programming of bit-image drawings.

Which just happens to be our very next topic.

5

How to Draw with Dots

NOW that we have our building materials—dot patterns, dot codes, control codes, and BASIC commands—we're ready to start drawing on the dot-matrix printer. In this chapter, drawing in general is the topic—from simple lines to textured drawings that resemble photographs.

Lines and Curves

By now it should be an easy matter to make the printer draw a straight line. All we have to do on a TRS-80 printer, after entering graphic mode, is select one of the single-dot codes and tell the printer how many times we want the printing of that dot repeated. If we want the line to be about two inches long, we have to repeat the printing 120 times in a printing density of 60 dots per inch. Suppose our selection is one of the middle dots, say, CHR\$(136). If we type `LPRINT CHR$(136);STRING$(120,136)` and then strike ENTER, the printer will immediately react by printing a line like the one in figure 5-1.

If we have an IBM PC driving an IBM or Epson printer, it isn't quite so easy: Printing 120 dots means we'll have to reserve 120 dot columns of graphic space first. Using the same density and a dot code of 16, we would need

FIGURE 5-1. Two-inch line, produced by repeating a single-dot code 120 times.

`LPRINT CHR$(27)"K"CHR$(120)CHR$(0);` to come before `STRING$(120,16)` to get the same result.

With an Apple II series computer controlling the Image-writer, the handy `STRING$` function probably won't be available. For a two-inch line, then, we'll need to reserve 160 dot columns (assuming a print density of 80 dots per inch) and repeat the `PEEK`-and-`POKE` routine 160 times (assuming Applesoft BASIC). For the Apple IIe, the following lines should do the job.

```

100 PR#1
110 PRINT CHR$(27)"N";
120 PRINT CHR$(28)"G0160";
130 FOR Z=1 TO 160
140 IF PEEK(49305) <> 16 THEN GOTO 140
150 POKE 49304,8
160 NEXT Z

```

This is a bit laborious by comparison. Programmers in printer graphics can only hope that the slogan "Apple II forever" doesn't mean Applesoft forever.

Normally, in `PRINT` and `LPRINT` statements, we need a trailing semicolon to prevent a line feed after each print of a dot when we are repeating a dot code, or else we'll get a vertical dotted line instead of a horizontal solid one. Notice, however, that the trailing semicolon is not needed after a `POKE` command (line 150 above). In fact, if you use one, the program will abort with a syntax-error message appearing on the display.

It is truly unfortunate that we have such an array of procedures for different systems just to print a two-inch line. We're stuck with this state of affairs, and to keep this book under a thousand pages, we'll have to forgo the practice of showing how all three printer families need to be programmed for each task. In most of the examples that follow, we will show the programming in detail for one type of printer and indicate how to translate for other printers. Usually a program for TRS-80 printers will be shown first, because programming is easier with those printers, and easier for the reader to follow as well.

Horizontal lines. Figure 5-2 shows a variety of horizontal lines and the BASIC commands for them with printers in the

HORIZONTAL LINES:BASIC Command



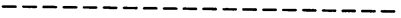





SOLID, THIN:		LPRINT STRING\$(200,136)
SOLID, THICK:		LPRINT STRING\$(200,156)
DASHED, THIN:		FOR N=1 TO 20: LPRINT STRING\$(7,136); STRING\$(3,128);: NEXT N
DASHED, THICK:		FOR N=1 TO 40: LPRINT STRING\$(4,158); CHR\$(128);: NEXT N
DOTTED, LIGHT:		FOR N=1 TO 100: LPRINT CHR\$(136); CHR\$(128);: NEXT N
DOTTED, HEAVY:		FOR N=1 TO 100: LPRINT STRING\$(2,140); STRING\$(2,128);: NEXT N
PATTERNED:		FOR N=1 TO 10: LPRINT STRING\$(12,140); STRING\$(2,128);STRING\$(4,140); STRING\$(2,128);: NEXT N
PATTERNED:		FOR N=1 TO 10: LPRINT STRING\$(5,156); CHR\$(128);CHR\$(156);CHR\$(128); CHR\$(156);CHR\$(128);: NEXT N

FIGURE 5-2. An assortment of horizontal lines and the BASIC commands for producing them with the DMP-400 printer.

Tandy family. The TRS-80 control code for entering graphic mode, CHR\$(18), is omitted in the list of commands.

Notice in figure 5-2's examples that to switch from a thin to a thick line, all you have to do is change from a single-dot code, such as CHR\$(136), to a multiple-dot code, such as CHR\$(156) or CHR\$(158). And if you want breaks in the line, for dotted, dashed, or patterned lines, just insert the blank code, CHR\$(128).

Figures 4-3 and 4-7 in the previous chapter can be used to translate the TRS-80 dot-codes into IBM-Epson or Apple codes. With the IBM-Epson and Apple printers, control codes for entering graphic mode and specifying density and dot-column space have to precede the print command in each case.

Vertical lines. Unlike horizontal lines, vertical lines require almost as many line feeds as dot prints. The line-feed size has to be set in accordance with the height of each print, so that adjacent lines of print will be butting. A feed of $\frac{7}{72}$ inch is automatic with seven-dot-high prints on the TRS-80







printers, but for the same height IBM-Epson and Apple printers would have to be preset to $\frac{7}{32}$ inch (use $\frac{14}{144}$ inch on the Imagewriter).

To make a single, thin vertical line on a TRS-80 printer, you can use a FOR-NEXT loop to repeat the code for all seven dots, CHR\$(255), followed by a line feed and carriage return. CHR\$(13) usually does both if the computer is also a TRS-80. If you plan to have the line located somewhere to the right of the left margin, a shorthand code for positioning, such as P\$=CHR\$(27)+CHR\$(16)+CHR\$(0)+CHR\$(100) for the 100th dot column, should precede each dot print. The examples in figure 5-3 show these line-feed and positioning features in all cases.

To make thicker lines, you merely repeat the dot code before each line feed, by using STRING\$, the 28+N sequence, or a second FOR-NEXT loop nested within the first. There is even a fourth way—LPRINT CHR\$(255);CHR\$(255);CHR\$(255);CHR\$(13) instead of a STRING\$(3,255) function inside a FOR-NEXT loop—but the STRING\$ function is obviously the simplest method.

Again, the IBM-Epson and Apple printers would have to make their graphic space reservations before printing, only now the enter-graphic sequence (27+“K”+N1+N2 or

FIGURE 5-3. Vertical lines produced by the DMP-400 printer.

SOLID, THICK:		FOR N=1 TO 4: LPRINT P1\$;STRING\$(3,255);CHR\$(13);: NEXT N
DOUBLE:		FOR N=1 TO 4: LPRINT P1\$;CHR\$(255);CHR\$(128);CHR\$(255);CHR\$(13);: NEXT N
DOTTED:		FOR N=1 TO 4: LPRINT P1\$;STRING\$(2,182);CHR\$(13);: NEXT N
DOTTED (FINE):		FOR N=1 TO 2: LPRINT P1\$;CHR\$(213);CHR\$(13);P1\$;CHR\$(170);CHR\$(13);: NEXT N
DASHED:		FOR N=1 TO 4: LPRINT P1\$;CHR\$(190);CHR\$(13);: NEXT N
DOT-AND-DASH:		FOR N=1 TO 4: LPRINT P1\$;CHR\$(222);CHR\$(13);: NEXT N

27 + "Gnnnn" respectively) has to be placed within the FOR-NEXT loop, even if there's only one dot column needing to be reserved. This takes some care, because if you reserve more dot columns than there are dot codes to fill them, the line-feed or carriage-return codes could be treated as dot codes and printed as dot patterns. With these printers, a combination line feed and carriage return can also be done with CHR\$(13) alone, provided that no trailing semicolon follows CHR\$(13) in the LPRINT or PRINT command.

Horizontal and vertical lines can also be made outside of graphic mode with punctuation marks or built-in line-graphics characters. A dashed horizontal line, for instance, can be made by repeating a hyphen, and a vertical line can be produced on the IBM Graphics or Radio Shack printers by repeating the text-mode code for a filled column (code 186 for the IBM, 245 for TRS-80s). Line feeds would have to be set to suit the heights of the characters ($\frac{9}{72}$ inch or $\frac{1}{8}$ inch on the IBM, $\frac{6}{72}$ or $\frac{1}{12}$ inch for the TRS-80s).

Vertical lines with ultra-high resolution. There's no law saying we always have to use the standard line feeds of graphic mode. Instead of the usual $\frac{7}{72}$ -inch or $\frac{8}{72}$ -inch feeds, we can use the small $\frac{1}{72}$ -inch or even the tiny $\frac{1}{144}$ -inch or $\frac{1}{216}$ -inch line spacings.

How big are $\frac{1}{216}$ -inch line feeds? Are they even visible? Would we ever really use such minuscule spacing very much? One way to get a handle on this ultra-high-resolution mode is to print a single dot and a horizontal line of dots and gradually increase the line feed from $\frac{1}{216}$ inch to, say, $\frac{9}{216}$ in single steps. This is what program 5-1, ULTHIRES, does.

PROGRAM 5-1. ULTHIRES. Varying the number of $\frac{1}{216}$ -inch and $\frac{1}{72}$ -inch line feeds in ultra-high-resolution line drawing.

```
10 'PROGRAM 5-1: "ULTHIRES" -- ULTRA-HIGH-RESOLUTION VERTICAL LINES
15 'FOR TRS-80 COMPUTERS AND PRINTERS
30 P1$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(100)      '100th DOT-COLUMN
100 FOR Z=1 TO 9
110   FOR N=1 TO 25
120     LPRINT CHR$(18);P1$;CHR$(129);STRING$(8,128);STRING$(20,129);
130     FOR S=1 TO Z: LPRINT CHR$(27);CHR$(51);: NEXT S      'Z/216" LINE FEED
140   NEXT N
150   LPRINT CHR$(13);                      'CARRIAGE RETURN + LINE FEED
160 NEXT Z
```

Line 120 of ULTHIRES prints the top dot (code 129 in the TRS-80 system) and a line of twenty such dots. The FOR-NEXT loops starting at lines 100 and 110 repeat the printing of the dot and line at a given line feed and increase the line feed after every twenty-fifth line of print. Line 130 governs the actual size of a line feed by controlling the number of repetitions (Z) of the $\frac{1}{216}$ -inch feeds that occur after each print line. Figure 5-4 shows the progression that results.

 $\frac{1}{216}$ " **$\frac{1}{72}$ "**

FIGURE 5-4. Printouts from the ULTHIRES program, showing changes in line feeds from $\frac{1}{216}$ inch to $\frac{9}{216}$ inch (left) and from $\frac{1}{72}$ inch to $\frac{3}{72}$ inch (right).

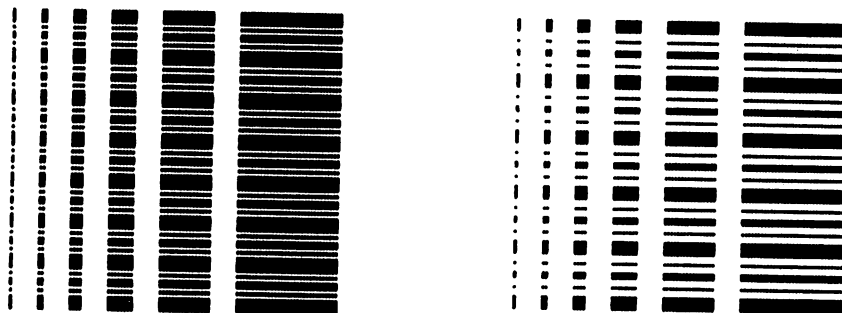
Notice that three repetitions of the $\frac{1}{216}$ -inch line feed amount to $\frac{3}{216}$ inch which reduces to $\frac{1}{72}$ inch. For every $\frac{1}{72}$ -inch feed there are three $\frac{1}{216}$ -inch feeds. To assure yourself that this three-to-one ratio actually comes out that way physically, change the control code in line 60 so that a $\frac{1}{72}$ -inch instead of $\frac{1}{216}$ -inch feed is repeated by **FOR S1 TO Z**. You should get what we have in the right-hand portion of figure 5-4: $\frac{3}{216}$ inch = $\frac{1}{72}$ inch, $\frac{6}{216}$ inch = $\frac{2}{72}$ inch, and $\frac{9}{216}$ inch = $\frac{3}{72}$ inch in the spacing of the dot and line prints.

Now we have some answers to our questions. A $\frac{1}{216}$ -inch line feed is so small that dots “separated” by such a tiny “space” overlap a great deal. In fact, they still overlap at $\frac{3}{216}$ inch, and seem to be touching each other at $\frac{3}{216}$ inch (depending on how consistent the line feed is) and $\frac{4}{216}$ inch (depending on how well-inked your ribbon is). Clear separation doesn’t come until $\frac{5}{216}$ -inch spacing.

Recall from earlier chapters that the size of a single dot is pretty well standardized in most printheads at $\frac{1}{72}$ inch. How big is a $\frac{1}{216}$ -inch line feed, then? A third of a dot. If this seems too small to be of much use, remember that in computer graphics you’re usually better off too small than not small enough—those “too small” things can be combined or repeated to make just the size you want. And as we’ll see in the next chapter, the ultrafine spacing of $\frac{1}{216}$ inch (or $\frac{1}{144}$ inch with the Apple printers) makes computerized drawing a whole new ball game.

If we add several more **FOR-NEXT** loops to **ULTHIRES** that repeat the horizontal dots or some blanks, we’ll have an example of some dot-and-dash vertical lines. To make it more interesting, however, let’s expand line 120 so that it prints

FIGURE 5-5. Patterned vertical lines of six different widths produced by $\frac{5}{216}$ -inch (left) and $\frac{3}{72}$ -inch (right) blank line feeds.

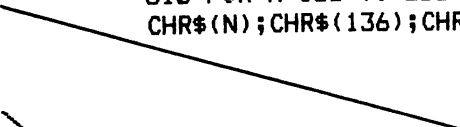


horizontal dot groups of 1, 2, 4, 8, 16, and 32. The printouts are in figure 5-5.

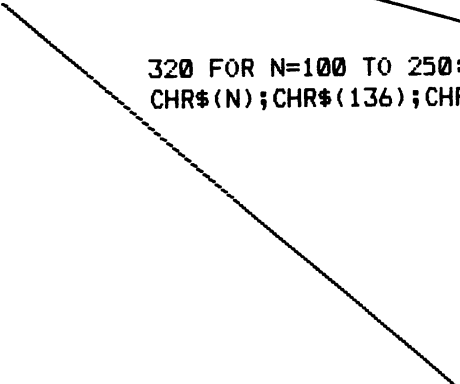
Oblique lines. Oblique (sloping or slanting) lines call for combinations of (vertical) line feeds and (horizontal) changes in the printhead's position. Figure 5-6 has four examples of such combinations for TRS-80 printers.

The first two examples show the difference in slope when $\frac{1}{72}$ -inch instead of $\frac{1}{216}$ -inch line feeds follow the printing of a single dot. The third example shows how an oblique dotted line is produced when the printing of a single dot is followed by a horizontal change of two dot columns and a vertical change of $\frac{5}{216}$ inch. The sloping dashed line in the fourth example illustrates the use of a running counter, $Y=Y+1$,

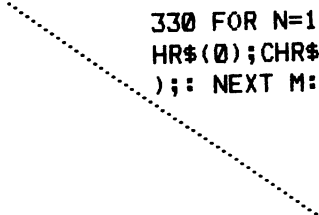
FIGURE 5-6. Oblique lines drawn by the DMP-400 printer.




```
310 FOR N=100 TO 250: LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);
CHR$(N);CHR$(136);CHR$(27);CHR$(51);: NEXT N
```



```
320 FOR N=100 TO 250: LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);
CHR$(N);CHR$(136);CHR$(27);CHR$(50);: NEXT N
```



```
330 FOR N=100 TO 200 STEP 2: LPRINT CHR$(18);CHR$(27);CHR$(16);C
HR$(0);CHR$(N);CHR$(136);: FOR M=1 TO 5: LPRINT CHR$(27);CHR$(51
);: NEXT M: NEXT N
```



```
340 Y=100: FOR N=1 TO 15: FOR M=1 TO 5: LPRINT CHR$(27);CHR$(16)
;CHR$(0);CHR$(Y);CHR$(136);CHR$(27);CHR$(51);: Y=Y+1: NEXT M: F
OR Q=1 TO 3: LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(Y);CHR$(128);
CHR$(27);CHR$(51);: Y=Y+1: NEXT Q: NEXT N
```

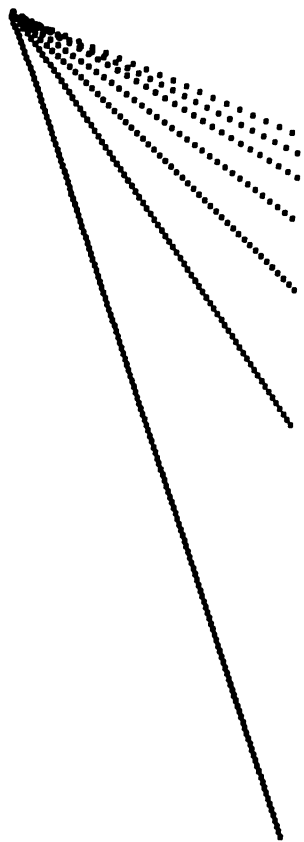



FIGURE 5-7. How the spacing of dots in oblique lines changes with their slope (from program 5-2, SLOPES).

inside two different FOR-NEXT loops, to control the positioning of more complex lines.

The biggest problem with oblique lines is how their dot spacing changes with their slope. Suppose we make a solid line with a steep slope by shifting to condensed (100 dots per inch) printing, a $\frac{2}{72}$ -inch line feed, and a bigger dot print composed of a box of four dots, using `STRING$(2,224)` instead of `CHR$(136)`. See how the steeply tilted solid line becomes a dotted line as the slope gets shallower (figure 5-7 and program 5-2).

You can go batty trying to make decent-looking solid lines stay that way when their slope changes a lot. When they are close to vertical, a box-type dot print produced by repeating a double-dot code once works fine, but as the slope decreases, a bigger dot print, such as the three-by-three `STRING$(3,156)`, will be needed to prevent blank space between the prints. And when they get closer to horizontal, you'll need to shift from a boxy to an elongated print, such as

PROGRAM 5-2. SLOPES. Varying the slope of oblique lines.

```

10 'PROGRAM 5-2: "SLOPES" -- OBLIQUE LINES (TRS-80 MODEL III, DMP-400)
20 LPRINT CHR$(30);CHR$(27);CHR$(20);          'CONDENSED PRINT DENSITY
100 FOR S=1 TO 7
110   FOR Y=100 TO 250 STEP S
120     LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(Y);STRING$(2,224);
130     PRINT Y
140     LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50);    '2/72" LINE FEED
150   NEXT Y
160   STOP      '(PAUSE WHILE PAPER IS SET BACK TO MARKED START POSITION)
170 NEXT S      '(AFTER TYPING 'CONT' TO RESUME PROGRAM)

```

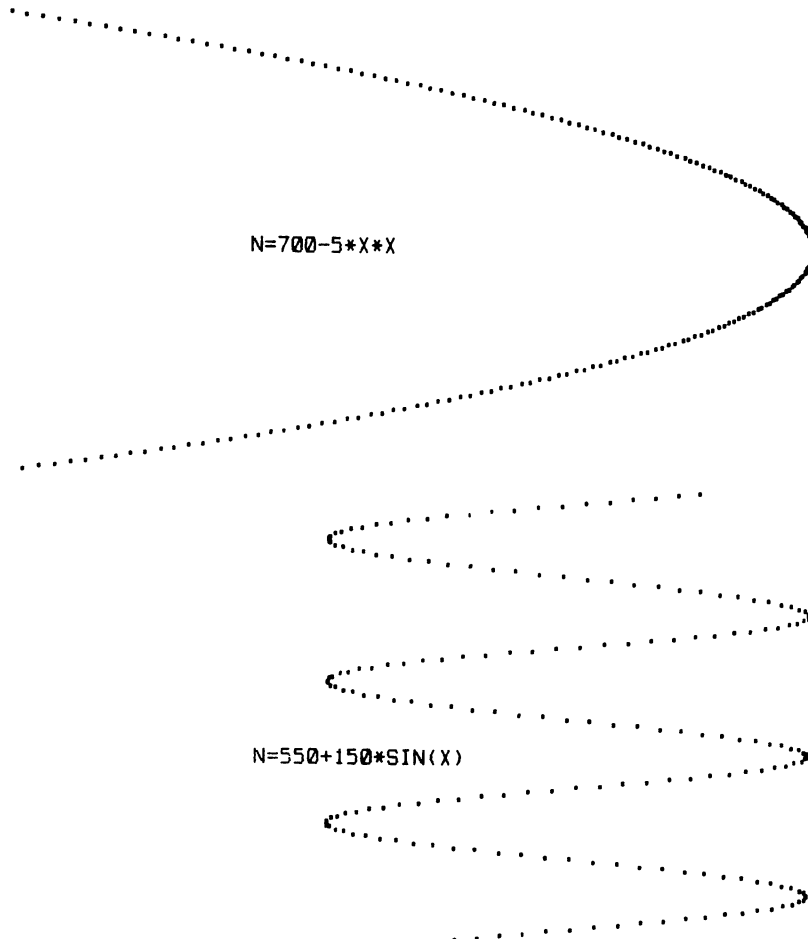
STRING\$(10,224), to keep them from being dotted or dashed. The result will be a visual mess, or at least a big disappointment compared to the way oblique lines look when they are properly drawn by hand in graphs.

If you want to keep things simple, stick to small dot prints for oblique lines and let them be dotted most of the time. If they turn into solid lines when they are steeply sloped, so be it. This is decidedly not the usual philosophy when it comes to oblique connecting lines in line graphs, however, and when we get to that topic in chapter 9, we'll see that we can get the computer and printer together to plot a dotted connecting line with nearly equal spacing of the dots regardless of its slope. That chapter will also take up the even more important topic of how to draw a line between any two specified points in a line graph.

In the BASIC statements presented in figure 5-6 and program 5-2, conversions from TRS-80 to IBM-Epson and Apple codes will be needed for reserving graphic space, using different dot codes, and using different control codes for line feeds and positioning. You can tell how much space to reserve from the FOR statements. Again, see chapter 4's conversion tables for the dot codes. Line feeds (without carriage returns) will need to be controlled by presetting codes and executed by CHR\$(10). Most importantly, the TRS-80's positioning sequence (27 + 16 + N1 + N2) has to be replaced by repeat-blank positioning on IBM-Epson printers, using STRING\$(N,0) in conjunction with graphic-space reservations and carriage returns, and by the 27 + "Fnnnn" positioning command on Apple printers.

Curved lines. Anytime that the *rate* of change in the print-head's horizontal position does not stay the same over successive, equal-sized line feeds, we will get a curved line instead of a straight one. In chapter 2, we saw how low-resolution curves could be produced by LPRINT TAB(N) commands with sine and cosine functions providing varying values of N (figure 2-7; programs 2-1 and 2-2). Here you can see how high-resolution curves can be generated by the 27+16+N1+N2 sequence (TRS-80), the 27+"K"+N1+N2+STRING\$(N,0) sequence (IBM-Epson), or the 27+"Fnnnn" sequence (Apple), again using sine functions for varying N or nnnn.

FIGURE 5-8. Bit-image curves drawn with mathematical functions on the DMP-400.



There's plenty of room for N (or nnnn) to vary if high densities are used in the bit-image programming. With the 15-inch TRS-80 DMP-400 in condensed printing, N can be any value from 0 to 1319. With the 9.5-inch Epson FX-80 in quadruple density, N can vary from 0 to 1919. With both of these printers, N values need to be converted to N1 and N2 values. With the 9.5-inch Apple Imagewriter, nnnn can vary from 0000 to 1280.

Two examples of curves, plotted one at a time, are in figure 5-8. These were plotted by the DMP-400 in 100-dots-per-inch density. The two equations stating how the positioning variable, N, varied with the number of small line feeds, X, are given in the figure. Considering the many built-in mathematical functions that even the lowest-priced home computers have—the DMP-400 was controlled by Radio Shack's little MC-10 in these two instances—there are endless variations in curve drawing that can be programmed easily.

You needn't restrict yourself to plotting only one function at a time. With a program like FUNCJAZZ (programs 5-3 and 5-4), you can amuse yourself for days drawing four curves at once.

PROGRAM 5-3. FUNCJAZZ/TRS. Drawing four mathematical functions simultaneously on TRS-80 printers.

```

10 'PROGRAM 5-3: "FUNCJAZZ/TRS" -- PLOTTING CONCURRENT MATH FUNCTIONS
20 'FOR TRS-80 PRINTERS AND TRS-80 MODEL III USING BOOTHE'S PRINTER DRIVER
30 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(27);CHR$(31); 'CONDENSED, BOLD
40 P$=CHR$(27)+CHR$(16)
50 LPRINT CHR$(18); 'ENTER GRAPHIC MODE
100 FOR X=-10 TO 10 STEP .1 ' (200 X VALUES)
110 A=500+30*SIN(X)+30*COS(2*X) '1ST FUNCTION
120 B=800-6*X*X '2ND FUNCTION
130 C=300+18*X+X^2 '3RD FUNCTION
140 D=1060-(20*X)*(25+X) '4TH FUNCTION
150 PRINT X,A;" ";B;" ";C;" ";D 'HELPFUL SCREEN INFO
160 GOSUB 230: LPRINT P$;CHR$(QA)+CHR$(AA);STRING$(3,142); 'PLOT A
170 GOSUB 240: LPRINT P$;CHR$(QB);CHR$(BB);STRING$(2,152); 'PLOT B
180 GOSUB 250: LPRINT P$;CHR$(QC);CHR$(CC);STRING$(2,134); 'PLOT C
190 GOSUB 260: LPRINT P$;CHR$(QD);CHR$(DD);STRING$(2,176); 'PLOT D
200 LPRINT CHR$(27);CHR$(50); '1/72" LINE FEED
210 NEXT X
220 END
230 QA=FIX(A/256): AA=FIX(A-256*QA): RETURN
240 QB=FIX(B/256): BB=FIX(B-256*QB): RETURN
250 QC=FIX(C/256): CC=FIX(C-256*QC): RETURN
260 QD=FIX(D/256): DD=FIX(D-256*QD): RETURN

```

PROGRAM 5-4. FUNCJAZZ/EPS. Concurrent math functions on the Epson FX-80.

```

10 'PROGRAM 5-4: "FUNCJAZZ/EPS" -- DRAWING MATH FUNCTIONS ON THE EPSON FX-80
20 'FOR TRS-80 MODEL III USING BOOTHE'S PRINTER DRIVER
30 LPRINT CHR$(27)"@"; 'MASTER RESET
40 GR$=CHR$(27)+"Z"
50 HOME$=CHR$(13)+CHR$(27)+"j"+CHR$(3) 'CARRIAGE RETURN
60 LPRINT CHR$(27)"3"CHR$(3); 'SET 3/216" LINE FEED
100 FOR X=-10 TO 10 STEP .1 ' (200 X VALUES)
110 A=500+30*SIN(X)+30*COS(2*X) '1ST FUNCTION
120 B=800-6*X*X '2ND FUNCTION
130 C=300+18*X+X^2 '3RD FUNCTION
140 D=1060-(20*X)*(25+X) '4TH FUNCTION
150 PRINT X,A;" "B;" "C;" "D 'HELPFUL SCREEN INFO
160 GOSUB 230: LPRINT CHR$(27)"D"CHR$(TA%)CHR$(0)CHR$(9);GR$;CHR$(XA+3);CHR$(0)
;STRING$(XA,0);STRING$(3,56);HOME$;
170 GOSUB 250: LPRINT CHR$(27)"D"CHR$(TB%)CHR$(0)CHR$(9);GR$;CHR$(XB+2);CHR$(0)
;STRING$(XB,0);STRING$(2,24);HOME$;
180 GOSUB 270: LPRINT CHR$(27)"D"CHR$(TC%)CHR$(0)CHR$(9);GR$;CHR$(XC+2);CHR$(0)
;STRING$(XC,0);STRING$(2,3);HOME$;
190 GOSUB 290: LPRINT CHR$(27)"D"CHR$(TD%)CHR$(0)CHR$(9);GR$;CHR$(XD+2);CHR$(0)
;STRING$(XD,0);STRING$(2,6);HOME$;
200 LPRINT CHR$(10);
210 NEXT X
220 END
230 TA%=FIX(A/24): XA=FIX(A-24*TA%): RETURN
250 TB%=FIX(B/24): XB=FIX(B-24*TB%): RETURN
270 TC%=FIX(C/24): XC=FIX(C-24*TC%): RETURN
290 TD%=FIX(D/24): XD=FIX(D-24*TD%): RETURN
300 RETURN

```

Program 5-3 is for TRS-80 printers and program 5-4 is for the Epson FX-80 in quadruple density. To guard against trouble codes, which are far more likely to be a problem in drawings of mathematical functions than in drawing situations where the user chooses the dot codes, both programs use Boothe's printer driver (program 3-4). Since the Epson printers don't have a direct-positioning command, a combination of horizontal tabbing in text mode (the 27+"D"+...+9 sequence) and fine tuning with repeated blanks in graphic mode is used in the four main print commands (lines 160-190). With IBM-Epson printers limited to double density, line 40 in program 5-4 should have its "Z" changed to "Y" and all the 24s in subroutines 230-290 should be changed to 12. For the range of X values used, the particular math functions chosen (lines 110-140) will stay within the margins of eight-inch printers whether they are plotted in

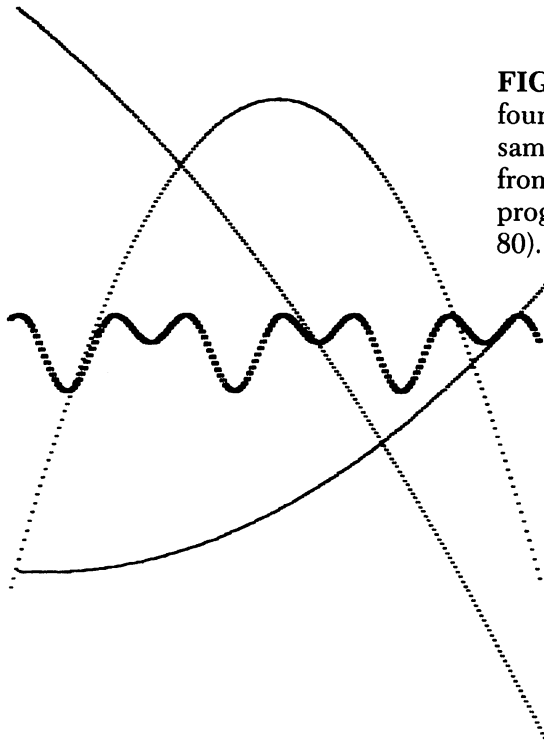


FIGURE 5-9. Plotting four different curves at the same time (FX-80 printouts from the FUNCJAZZ program for the Epson FX-80).

double or quadruple density. Figure 5-9 shows the FX-80 printout.

If Boothe's changes in the printer-drivers for the TRS-80 Model I or III are not executed before the FUNCJAZZ/TRS program is run, or if the Model 4 or Model 100 is driving the printer, detour statements will need to be added to the four subroutines, to dodge around problem codes (see chapter 3). In plotting curves, you'd be well advised to use the form of detour that involves incrementing and backspacing, or else the curves will be jagged.

Fooling around with multiple math functions this way is good practice for programming line graphs having several curves. For plotting different functions from those in the FUNCJAZZ programs, you can simply change the equations in lines 110–140. The range of X values can be changed just by altering what follows `FOR X=` in line 100, and you can vary the slopes of the curves by changing the number of times a $\frac{1}{72}$ -inch or $\frac{1}{216}$ -inch line feed occurs between prints by wrapping the line-feed command of line 200 in a `FOR-NEXT` counter loop.

Mathematical Forms and Designs

You can also use math functions to draw recognizable forms, such as circles and ellipses, spirals, flower- or butterfly-shaped figures, and "op art" creations.

Circular forms. In the Cartesian rectangular coordinate system (which we've been using in these recent examples), locations of points on a curve are given by combinations of X (horizontal) and Y (vertical) numbers. If we let R stand for the radius of a circle, i.e., the distance between the center of the circle and any point on its circumference, the equation for a circle can be stated simply:

$$X^2 + Y^2 = R^2$$

And if we solve this equation for Y (unsquared), we will get:

$$Y = \pm \sqrt{R^2 - X^2}$$

In BASIC the right-hand side of this equation would be translated to positive and negative values of $\text{SQR}(R*R - X*X)$. To plot a circle on the Model III's screen, you need only to vary X from the negative value of the radius, R, to its positive value, in small steps, and let X and Y vary in a pair of SET(X,Y) statements.

PROGRAM 5-5. XYCIRCLE. Circle drawing with rectangular coordinates.

```

10 'PROGRAM 5-5: "XYCIRCLE" -- DRAWING CIRCLES OR ELLIPSES
20 'FOR TRS-80 PRINTERS AND TRS-80 MODEL III (USING BOOTHE'S PRINTER DRIVER)
30 CLS
40 R=30                                'R=RADIUS OF CIRCLE
100 FOR X=-R TO R STEP .1
110  Y=SQR(R*R-X*X)                    'SOLVE CIRCLE EQUATION FOR Y
120  SET(64+1.2*X,24+.75*Y)            'PLOT OF CIRCLE ON SCREEN
130  SET(64+1.2*X,24-.75*Y)            '      "      "      "      "
140  REM -- INSERT 'GOTO 200' HERE IF ONLY SCREEN DUMP IS DESIRED
150  Y1=.9*(100+3*Y): QY=FIX(Y1/256): YY=FIX(Y1-256*QY%)
160  Y2=.9*(100-3*Y): QZ=FIX(Y2/256): ZZ=FIX(Y2-256*QZ%)
170  LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(QY);CHR$(YY);CHR$(136);
180  LPRINT CHR$(27);CHR$(16);CHR$(QZ);CHR$(ZZ);CHR$(136);
190  LPRINT CHR$(27);CHR$(51);          '1/216" LINE FEED
200 NEXT X
205 REM -- INSERT 'GOSUB 220' HERE IF SCREEN DUMP IS DESIRED
210 END

```

```

220 '*** DUMP FROM SCREEN TO PRINTER'S BIT-IMAGE BLOCKS (FROM PROG. 4-7) ***
230 DEFINT A-Z: DIM A(128,48)
240 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18); 'CONDENSED PRINTING
250 FOR ROW=0 TO 47
260   FOR COL=0 TO 127
270     A(COL,ROW)=-(POINT(COL,ROW))
280   NEXT COL
290   LPRINT CHR$(30);ROW;
300 NEXT ROW
310 DC=143: SZ=4
320 LPRINT STRING$(3,13);           '3 SPACES
330 FOR ROW=0 TO 47
340   FOR COL=0 TO 127
350     IF A(COL,ROW)=0 THEN LPRINT CHR$(18);STRING$(SZ,128);: GOTO 380
370     IF A(COL,ROW)>0 THEN LPRINT CHR$(18);STRING$(SZ,DC);
380   NEXT COL
390 FOR Z=1 TO SZ: LPRINT CHR$(27);CHR$(50);: NEXT Z      'LINE FEED
400 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(0);    'CARRIAGE RETURN
410 NEXT ROW
430 RETURN

```

Program 5-5 (XYCIRCLE) plots circles or ellipses on the Model III screen and DMP-400 printer simultaneously. In this program the screen plotting is handled by lines 110–130 and the printer plotting by lines 150–190. Both sets of lines are inside a FOR-NEXT loop that varies X from –30 to +30 in 600 steps. Whereas a new pixel lights up on the screen only every third or fourth step on the average, nearly every one of the 600 steps produces a print in a new location on the printer paper—this is the difference between low and high resolution.

Figure 5-10 shows both plots. Although the left-hand drawing was obtained by a screen dump, the printer plot on the right was not. The mathematical functions produced one set of X and Y values for the screen drawing and another set of Y (actually YY and ZZ) values for the printer, while the printer paper moved one line feed for each X value. As long as the plotting on the printer was left to right all the way (X varied from smallest to largest values), there was no problem.

We would have gotten an ellipse instead of a circle if we had multiplied Y values by some factor other than 3 in lines 150 and 160. (Adding and subtracting $3*Y$ from 100, by the way, made all values of Y1 and Y2 positive so they would be accepted by the printer.) We could also have made the circle

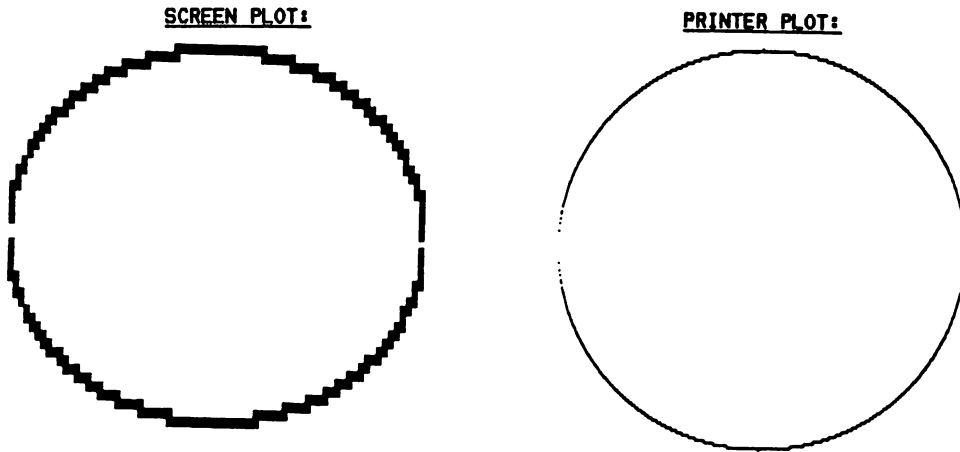


FIGURE 5-10. Drawing a circle on the TRS-80 Model III screen (left) and on the DMP-400 printer (right) using rectangular coordinates (XYCIRCLE program).

more elliptical by printing in elite or condensed pitches instead of the standard density of 60 dots per inch.

Changes to be made in converting this TRS-80 version of XYCIRCLE to an IBM-Epson version can be determined by comparing the previous FUNCJAZZ programs (programs 5-3 and 5-4). For the Imagewriter, the smallest line feed available is $\frac{1}{144}$ inch and the values of the multiplying constants in lines 150 and 160 would have to be changed markedly because of the difference in line feed. Further changes, of course, would be needed for the screen plotting on IBM and Apple computers.

Using polar coordinates. By and large, circular figures are easier to draw using polar coordinates instead of Cartesian rectangular coordinates. Whereas rectangular coordinates give the point's location by telling how far it is from the X and Y axes, polar coordinates tell how far the point is from the center of a circle and what size of angle the line connecting that point to the center makes with the horizontal line lying immediately to the right of the center. (If you're getting lost, don't panic—examples are coming, and it's not necessary to follow all this in order to use the programs).

Some marvelous forms can be created with polar coordinates, and these drawings provide some of the most stunning contrasts between printer plots and low-resolution screen drawings. Compare the samples in figure 5-11, for example.

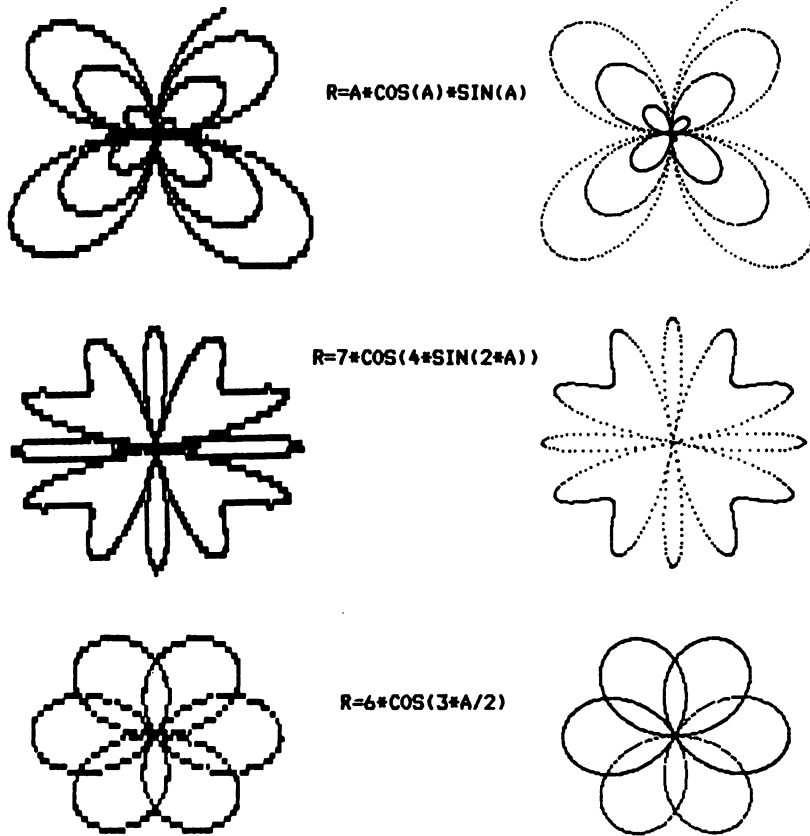
SCREEN PLOT:**PRINTER PLOT:**

FIGURE 5-11. Comparison of Model III screen dumps (left) and DMP-400 printer plots (right) using polar coordinates in the POLARRAY program.

In getting the printer to draw figures like these, we've crossed two important boundaries. Besides shifting to polar coordinates, we no longer have X values running nicely in order from left to right for the plotting. If you run the program for this below, you'll see the center of the figure light up first on the computer screen, and the drawing will continue on the screen going left, up, right, down—every which way in many loops. But the printer can't go left except by reverse line feed, and using that for every move to the left would be a programmer's nightmare. In fact, if you were limited to 1/6-inch and 1/12-inch reverse line feeds, as on the Radio Shack printers, the programming would be an impossible task.

So how did the printer (a TRS-80 LPVIII) do the drawings in figure 5-11? Again the answer is: not by screen dump. Both the screen and printer drawings were produced by a program called POLARRAY (program 5-6), for "polar array," and "array" is the key word here.

PROGRAM 5-6. POLARRAY. Screen and printer drawings of math functions using polar coordinates.

```

10 'PROGRAM 5-6: "POLARRAY" WITH BOTH SHELL-METZNER & CMD "0" SORTS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 PRINTERS AND MODEL III (USING BOOTHE'S DRIVER)
30 CLEAR 10000
40 CLS: LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(27);CHR$(31); 'CONDENSED, BOLD
50 AA=0: ZZ=12.6: SS=.03 'START, FINISH, STEP SIZE
60 ST=INT((ZZ-AA)/SS)+1 'NUMBER OF STEPS (PLOTTED POINTS)
70 DIM Z$(ST),Y(ST) 'SET ARRAY SIZES
100 FOR A=AA TO ZZ STEP SS 'VARIATION OF ANGLE (A)
110 R=A*COS(A)*SIN(A) 'RADIUS (R) AS A FUNCTION OF ANGLE
120 X=64+6*R*COS(A): Y=24+2*R*SIN(A) 'CONVERSION TO X-Y COORDINATES
130 H=INT(10*(X+100)): V=INT(10*(400-4*Y)) 'CONVERSIONS FOR BIT-IMAGE PRINTING
140 Z$(I)=STR$(H)+STR$(V) 'STORAGE OF H & V IN STRING ARRAY
160 SET(X,Y) 'PLOTING OF FIGURE ON SCREEN
170 I=I+1 'INCREMENT Z$(I) ARRAY SUBSCRIPT
180 NEXT A 'GO TO THE NEXT SIZE OF ANGLE (A)
190 GOSUB 4000 'CHANGE TO 'GOSUB 1000' FOR SHELL-METZNER
200 GOSUB 3000 'PRINTING OF FIGURE ON PRINTER
210 END

1000 '*** SHELL-METZNER SORT OF Z$(I) VALUES FOR COMPUTERS LACKING CMD"0" ***
1010 CLS: PRINT "SORT STARTING AT ";TIME$
1020 P=0
1030 M=ST
1040 M=INT(M/2)
1050 IF M=0 THEN GOTO 1230
1060 P=P+1
1070 FOR BL=0 TO M-1
1080 I=BL
1090 J=BL+M
1100 SW=0
1110 IF Z$(I)<=Z$(J) GOTO 1160
1120 SW=1
1130 B#=Z$(I) 'IF BASIC CONTAINS 'SWAP' COMMAND,
1140 Z$(I)=Z$(J) 'THESE THREE LINES CAN BE CHANGED
1150 Z$(J)=B# 'TO 'SWAP Z$(I),Z$(J)'
1160 I=J
1170 J=J+M
1180 IF J<ST GOTO 1110
1190 IF SW=0 GOTO 1210
1200 GOTO 1080
1210 NEXT BL
1220 GOTO 1040

```

```

1230 PRINT "SORT DONE AT ";TIME$
1240 RETURN
2000 '***** CONVERSIONS AND DETOUR FOR Y(I) VALUES *****
2010 QY=FIX(Y(I)/256): YY=FIX(Y(I)-256*QY)
2020 IF INT(YY)=9 THEN ZZ=YY+1: QZ=QY: LPRINT CHR$(27);CHR$(16);CHR$(QZ);CHR$(ZZ
);CHR$(30);CHR$(8);CHR$(2);CHR$(18);CHR$(136);: ZZ=0: QZ=0: GOTO 2040
2030 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(QY);CHR$(YY);CHR$(136);
2040 RETURN
3000 '***** PRINTING AFTER SORTING & CONVERSIONS *****
3010 KM=1095
3020 K=1100
3030 FOR I=0 TO ST-1
3040 IF VAL(LEFT$(Z$(I),5))>K THEN LPRINT CHR$(27);CHR$(51);CHR$(27);CHR$(51);
: KM=K: K=K+5: GOTO 3040
3050 IF VAL(LEFT$(Z$(I),5))>KM AND VAL(LEFT$(Z$(I),5))=<K THEN PRINT Z$(I): Y(
I)=VAL(RIGHT$(Z$(I),4))/10-100: GOSUB 2010
3060 NEXT I
3070 RETURN
4000 '***** TRS-80 MODEL III MACHINE-LANGUAGE RAPID SORTING *****
4010 I%=ST: CMD "O", I%, Z$(0)
4020 RETURN

```

Lines 100–120 of POLARRAY generate the X and Y values that are plotted on the Model III screen by line 160's PRINT@ (Model 4) or SET (Model III) statement. Using polar coordinates, line 100 provides for varying the angle (A) over 420 small steps, and line 110 states the mathematical function by which the radius (R) varies with the angle. (If the radius does not vary with the angle, that is, if R is a constant, the program draws circles or ellipses.) Line 120 translates the polar coordinates, R and A, into rectangular coordinates, X and Y, which are needed by both the computer and the printer for plotting.

As the 420 pairs of X and Y values are fed into the SET(X,Y) drawing statement, the figure is plotted on the screen while the printer remains silent. During the screen drawing, all of the X and Y values are converted to string-type values, linked together (concatenated by a + sign), and stored in a string array labeled Z\$(I). The array storage is provided by line 70's DIM statement, the definitions of H (for "horizontal") and V ("vertical") in line 130, and the conversion to linked strings in line 140.

The X values are converted to four-digit integers in line 130 to make it easier to sort them in a later stage. The Y values have to be transformed into numbers that are in keep-

ing with the printer's numbering system for dot columns (multiplying by 4 does that) and subtracted from some constant (400 in this case) that is greater than the highest Y value, to avoid negative values. This subtraction also makes up for the fact that Y values for screen plotting are "upside down"—they range from 0 at the top of the screen to 48 at the bottom. Hence line 130's $400 - 4 * Y$. The counter in line 170, $I = I + 1$, provides the subscript for the $Z(I)$ array.

Here's the main trick for the plotting by the printer: After all the X and Y values have been stored, the converted X values are sorted from lowest to highest, and then the converted Y values are plotted in order of the X values. That way, the printer's plotting proceeds in the usual left-to-right fashion; no backward line feeds are needed.

The sorting is handled with dispatch by subroutine 4000, which makes use of the Model III's wonderful utility called CMD "O" in line 4010. This machine-language feature, which can only sort one-dimensional string arrays, takes less than two seconds to sort the 420 values of $Z(I)$ on the basis of the ASCII codes of the first four digits (the "X component") of each $Z(I)$ value. The order of the $Z(I)$'s is of course changed by the sorting, but because of the linkage of the "Y component" to the "X component," any information about Y values is carried along with X-axis information in the rearrangement of the array values.

When it comes to printing the reordered array (subroutines 2000 and 3000), the Y-axis values in the right-hand half of each $Z(I)$ value are "unstrung" in line 3050, that is, converted back to numerical values by VAL and RIGHT\$ functions. As the subscript (I) is varied from 0 to 419 (and as a result, X values are varied from lowest to highest) by line 3030, the Y values linked to each of the X values are rapidly printed in subroutine 2000. Line feeds ($\frac{3}{2}_{16}$ inch) occurring at equal steps along the X axis are provided by line 3040 in conjunction with values of K and KM, which define the limits of a continually changing, narrow band of X values. As the narrow band travels over the entire range of X values, it alertly sniffs out all values of X that fall within the band, and after their linked Y values are printed, it moves on to the next position in the series.

Computers not offering the handy CMD "O" utility for

sorting can sort by slower methods in BASIC, or even use the moving-narrow-band technique for searching instead of sorting. But if you have several hundred X-Y pairs to deal with, the search method may take over an hour. The best-known sorting technique in BASIC—the “bubble sort” method—may take over half an hour (more or less, depending on the number of pairs and their original order). The faster Shell-Metzner sort, however, will bring the sorting time down to a few minutes, particularly if computers faster than the Model III are used. Subroutine 1000 of the POLARRAY program uses this latter technique, providing a decent alternative to the CMD “O” method. For Model 4 users, there is now a method that is just about as fast as CMD “O,” thanks to Alan D. Smith (*80 Micro*, March 1985).

POLARRAY is a versatile program for both screen and printer plotting of circular figures that are calculated by polar coordinates. For different figures—spirals, cardioids, flowers, many others—it’s mostly a matter of changing the function in line 110. For some snazzy ones, David A. Kater and Susan J. Thomas (*TRS-80 Graphics for the Model I and Model III*, BYTE Publications, 1982) suggest trying $R = .3 * A$, $R = b * \cos(\sin(b * A))$, $R = b * \cos(A) + 2$, or $R = b * \cos(2 * \sin(A))$. With large numbers of Z\$(I) values, say, over a thousand, you may run into further delays (up to several minutes) when the computer takes time out to reorganize its string storage in RAM; a generous provision for clearing string space, such as `CLEAR 20000`, will reduce the string-reorganization time somewhat.

An IBM-Epson version of POLARRAY would require many changes in program 5-6, even those lines in the program concerned with mathematical manipulations rather than printing procedures. In the PC’s high-resolution graphics mode the pixels would be in a 200-by-640 matrix as opposed to the TRS-80 Model III’s 48-by-128, so the main changes for the screen plotting would be the addition of `SCREEN 2`, using `PSET(X,Y)` instead of `SET(X,Y)`, and a change in line 120 to $X = 320 + 21 * R * \cos(A) : Y = 100 + 9 * R * \sin(A)$.

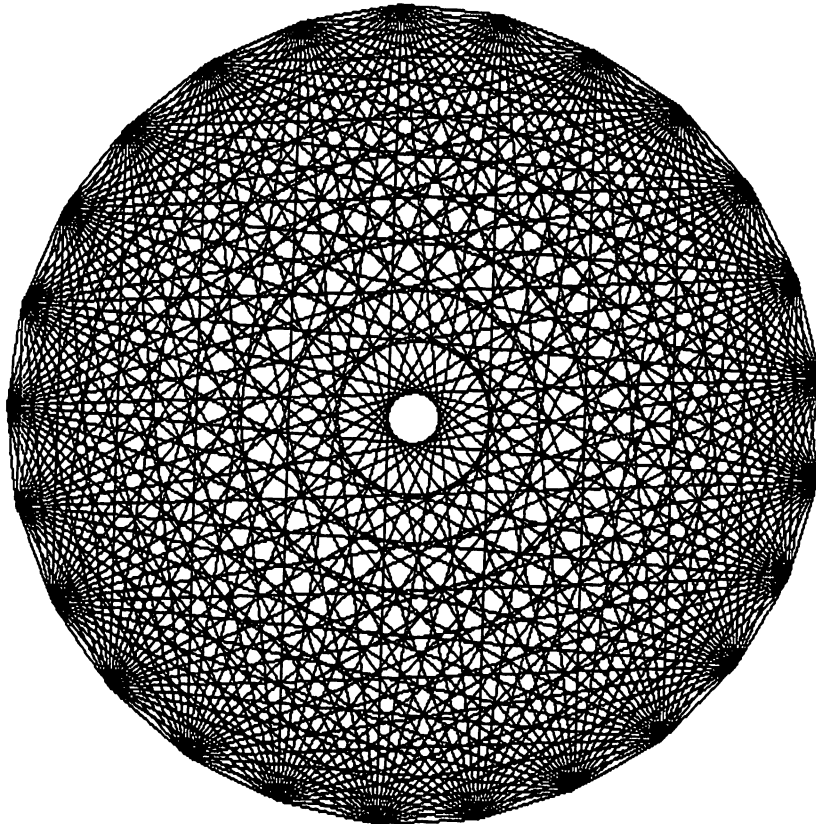
In the printer’s plotting, the usual `OPEN`, `WIDTH`, and `PRINT#` should be used, along with double density. If positioning is done with a combination of horizontal tabbing and

repeat-blank movement, line 2020 should be deleted and lines 2010 and 2030 changed to:

```
2010 HT=FIX(Y(I)/12):GR=FIX(Y(I)-12*HT)
2030 PRINT #1, CHR$(27)"D"CHR$(HT)
      CHR$(0)CHR$(9)CHR$(27)"Y"CHR$(GR+1)
      CHR$(0);STRING$(GR,0);CHR$(8)
```

Complex math-generated figures. At higher levels of both mathematics and drawing, you can find several excellent programs in the microcomputer literature. One of the best is the NETWORK program presented by Delmer Hinrichs in *80 Micro* (May 1983). Figure 5-12 illustrates the kind of drawings his program does on Epson printers in both BASIC and FORTRAN. The BASIC programs take up to sixteen minutes to print (compiled BASIC would reduce that by 75%), and

FIGURE 5-12. One of the many different printouts produced by Hinrichs's NETWORK program for IBM-Epson printers.



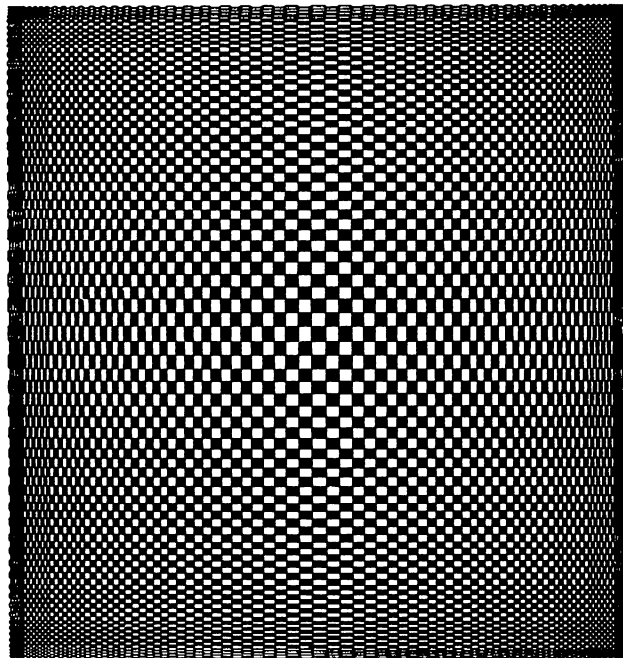
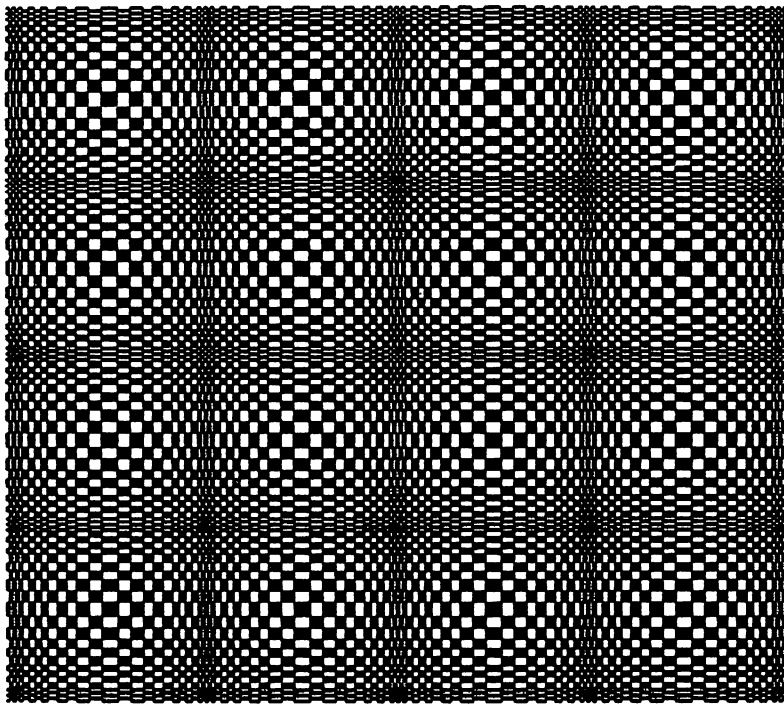


FIGURE 5-13. Printouts on the DMP-400 from the TRS-80 version of the OPTICART program. (Based on Kater's SYMMETRY program in the Epson FX-80 user's manual, used with permission of Epson America, Inc.)

additional time for calculations, depending on the number of points or lines in a figure.

By answering a few INPUT prompts, the user can draw pointed stars or polygons (in which the points of a star are joined by straight lines), and either of these can be enclosed

within a circle. The number of points can vary up to fifty, although figures having more than twenty-five points require a lot of time. If there is an odd number of points, the figure's center will be open, as in figure 5-12; with an even number, the center will be closed (by lines drawn through it). Only the last five lines of Hinrichs's BASIC programs need to be changed for printers other than the IBM Graphics or Epsoms.

Equally fascinating is the SYMMETRY program by David Kater in the Epson FX-80 manual (pp. 183-193). Using this program, you can produce printouts like the "op art" creation in figure 5-13. Kater cleverly generates a two-dimensional pattern from a one-dimensional array, and his program lets you vary the number of repeated squares within the total pattern and size ratios within the squares. Although designed for the FX-80, his program will produce even more remarkable designs and optical effects if it is modified to run in bold-printed graphics on a TRS-80 or Apple printer (Figure 5-13 is a printout by the DMP-400). The TRS-80 version, called OPTICART, is program 5-7.

PROGRAM 5-7. OPTICART. "Op art" drawing by a printer (Epson-to-Tandy conversion of Kater's SYMMETRY program for the FX-80).

```

10 'PROGRAM 5-7: "OPTICART" -- EPSON-TO-TANDY CONVERSION OF KATER'S 'SYMMETRY'
20 LPRINT CHR$(30);CHR$(27);CHR$(31);
30 DIM A(480): X=1: C=0
40 MAX=5: MIN=1: RE=4: N=0
50 FOR J=1 TO RE
60   N=N+1
70   GOSUB 1000
80   IF N<MAX THEN 60
90   N=N-1
100  GOSUB 1000
110  IF N>MIN THEN 90
120 NEXT J: PRINT
130 FOR K=1 TO C: PRINT A(K);: NEXT K: PRINT: PRINT "C="C
140 LAST=INT(C/7): R=C-7*LAST
150 FOR PASS=0 TO LAST: P=0: PRINT "PASS "PASS" OF "LAST
160 H=6: IF PASS=LAST THEN H=R-1
170 FOR DOT=0 TO H
180  IF A(7*PASS+DOT+1)=1 THEN P=P+2^(6-DOT)
190 NEXT DOT
200 GOSUB 5000      'CONVERT EPSON TO TRS-80 CODE
210 P0=127-NTRS: IF PASS=LAST THEN P0=P0+1-2^7-R
220 PRINT "P0=";P0
230 LPRINT CHR$(18);
240 FOR K=1 TO C

```

```

250 IF A(K)=1 THEN LPRINT CHR$(NTRS+128);
260 IF A(K)<>1 THEN LPRINT CHR$(P0+128);
270 NEXT K
280 LPRINT
290 NEXT PASS
300 END
1000 FOR K=0 TO MAX-N
1010 FOR L=1 TO N
1020 C=C+1: A(C)=X
1030 NEXT L: X=1-X
1040 NEXT K: PRINT N;: RETURN
5000 'CONVERSION OF EPSON TO TRS-80 DOT CODES
5010 NEPS=P
5020 FOR E=6 TO 0 STEP -1 ' (7 PINS, NUMBERED 6 TO 0)
5030 X(E)=NEPS-2^E ' (CONVERSION OF
5040 IF X(E)<0 THEN Y$(E)="0": GOTO 5060 ' DECIMAL CODE
5050 Y$(E)="1": NEPS=X(E) ' TO BINARY FORM)
5060 NEXT E
5070 NTRS=64*VAL(Y$(0))+32*VAL(Y$(1))+16*VAL(Y$(2))+8*VAL(Y$(3))+4*VAL(Y$(4))+2*
VAL(Y$(5))+VAL(Y$(6)) ' CONVERT TO MIRROR-IMAGE DECIMAL (TRS-80 CODE)
5080 RETURN

```

Drawing Pictures

If you supply enough information to the printer, it will draw almost anything you want. But pay attention to those words “enough information” and get ready for some hard labor, because now we’re talking about drawing things that can’t be described by mathematical functions. This means we won’t have the computer calculating most of the dot codes that are sent to the printer—somehow we have to come up with the codes ourselves.

Simple line drawings. If we wanted to draw a curvy line with no particular mathematical form to it, we could just send a series of single-dot codes to fire pins that are next to each other, such as (on the IBM) 8, 4, 4, 2, 1, 2, 4, 8, 16, etc. In an actual program, we would have something like this:

```

10 LPRINT CHR$(27)"K"CHR$(21)CHR$(0)
   'RESERVE 21 GRAPHIC COLS.
20 LPRINT CHR$(16)CHR$(32)CHR$(32)CHR$(64)
   CHR$(128)CHR$(64)CHR$(32)CHR$(16)CHR$(8)
   CHR$(8)CHR$(4)CHR$(8)CHR$(8)
   CHR$(4)CHR$(2)CHR$(2)CHR$(4)
   CHR$(8)CHR$(4)CHR$(8)CHR$(16);

```

and it would print something like the line in figure 5-14.



FIGURE 5-14. Wavy line produced by a series of twenty-one single-dot codes.

For all that typing, a pretty skimpy output! If only we didn't have to keep repeating `CHR$()`! Well, we wouldn't have to if we used `READ` and `DATA` statements—then we could just list the dot codes (DCs) in `DATA` lines and tell the computer to `READ DC` and `LPRINT CHR$(DC)`. But even with that timesaver it'll usually take over five hundred dot codes just to draw a simple figure only an inch high.

You can get an idea of how bad it is from the jack-o'-lantern drawing in figure 5-15. This illustration shows the DMP-400's printout in actual size from a TRS-80 program that sends 630 DC values to the printer (nine print lines, each having seventy codes). A few lines of the DC values themselves are shown under the drawing.

This seems to be a line of work for people who like to hook rugs, read timetables, or break world records for domino chains. But even for those patient souls, there must be a better way.

Again, there is. We could use the binary sum (BS) codes—on the average, they are smaller numbers that take up less memory space than the three-digit codes of TRS-80 printers—and change our print command from `LPRINT CHR$(DC)` to `LPRINT CHR$(BS+128)`. Also, every time we wanted to repeat a dot code, we could have it preceded by another number telling it how many times to repeat, instead of having to type the same code number over and over. We could also use a special number (higher than any of the dot-code numbers) for telling the printer when to start a new line.

If we switched our DCs to BSs, used negative numbers for repeating, and used code 999 for a line feed and carriage return, all nine of our `DATA` lines for the jack-o'-lantern would fit in the space that only the first few lines in figure 5-15 take up. That's progress, but the program that runs these new `DATA` lines has to be designed to handle the repeating and end-of-line events as well as the `LPRINT` commands. It will need a distinction between how it treats a dot code that is preceded by a negative number and how it treats one that



128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	192	192	192	192	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
176	136	132	130	129	129	128	240	136	132	131	129	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
192	192	192	160	160	160	160	160	160	160	160	160	160	192	252	195
192	192	192	192	192	192	255	192	160	160	160	160	160	160	160	160
192	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	224	144	136	132	130	129
129	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	129	130	132	136	144	160	192	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128	240	135	128	128	128	128	128	128
128	128	128	128	128	128	152	164	194	129	193	194	164	148	136	128
128	128	128	128	128	128	128	152	164	194	129	194	162	148	136	128
128	128	128	128	128	128	128	128	129	134	184	192	128	128	128	128

That negative value of N is converted to a positive value by line 140's `LPRINT STRING$(-N,128+M)`—minus a minus is a plus. Placed before the comma in `STRING$`, the converted N value gives the number of times the M code (in-

PROGRAM 5-8. HALOWEEN. Freehand drawing of a jack-o'-lantern.

```

10 'PROGRAM 5-8: "HALOWEEN" -- JACK-O'-LANTERN DRAWING
20 'FOR TRS-80 COMPUTERS AND PRINTERS (USE BOOTHE'S PRINTER DRIVER IF MOD. III)
30 LPRINT CHR$(30);CHR$(27);CHR$(23);CHR$(18); 'ELITE DENSITY, GRAPHIC MODE
100 FOR R=1 TO 9 'LOOP FOR 9 ROWS
110 READ N: IF N=999 THEN GOTO 150 'READ CODE #, LINE FEED IF 999
120 IF N>=0 AND N<128 THEN LPRINT CHR$(128+N);: GOTO 110 'PRINT DOT-PATTERN
130 READ M 'CODE WAS NEGATIVE, SO READ NEXT AND CALL IT M
140 LPRINT STRING$(-N,128+M);: GOTO 110 'REPEAT-PRINT DOT-PATTERN
150 LPRINT CHR$(13); 'LINE FEED & CARRIAGE RETURN
160 NEXT R
170 END
210 DATA -38,0,-4,64,-28,0,999
220 DATA -31,0,64,48,8,4,2,1,1,0,112,8,4,3,1,-26,0,999
230 DATA -17,0,64,64,-10,32,64,124,67,-6,64,127,64,-8,32,64,-21,0,999
240 DATA -10,0,96,16,8,4,2,1,1,-32,0,1,2,4,8,16,32,64,-14,0,999
250 DATA -8,0,120,7,-12,0,24,36,66,1,65,66,36,20,8,-8,0,24,36,66,1,66,34,20,8,-9
,0,1,6,56,64,-10,0,999
260 DATA -8,0,127,-16,0,1,-6,0,96,24,6,6,24,96,-20,0,96,31,-10,0,999
270 DATA -8,0,1,14,112,-7,0,4,8,24,40,-3,80,32,32,96,0,64,64,67,66,66,2,-3,66,67
,64,64,0,96,32,32,16,80,8,72,36,28,6,2,-3,0,96,30,1,-11,0,999
280 DATA -11,0,7,24,32,32,64,-8,0,2,5,4,-3,9,12,4,28,19,18,19,16,28,4,28,17,17,9
,8,4,3,0,1,-3,0,64,32,16,12,3,-14,0,999
290 DATA -16,0,1,1,-3,2,-7,4,-10,8,-8,4,-3,2,1,1,-19,0,999

```

creased by 128) is to be repeated. If the $28 + N$ sequence had been used for repeating instead of the `STRING$` function, line 140 would have been `LPRINT CHR$(28);CHR$(-N);CHR$(M+128);:GOTO 110`. If `FOR-NEXT` had been used, the loop for repeating would have started with `FOR X=1 TO -N`.

If we substitute another set of `DATA` statements for those in `HALOWEEN` and adjust the `FOR-NEXT` statement in line 100 to accommodate a larger number of print lines, we'll get a different drawing. The cartoon drawing of Dennis the Menace in the first chapter (figure 1-2) was done using a program that was nearly identical to the `HALOWEEN` program, but with twenty-nine `DATA` statements instead of nine and twenty-three print lines instead of nine (thus line 100 was `FOR R=1 TO 23`). If you want to type in the `DATA` lines and see for yourself, you'll find all the codes for the cartoon in appendix A (program A-1, "DENNIS").

With a program like `HALOWEEN`, you're not restricted to bare line drawings that have no shaded or darkened areas. Another set of codes in place of the jack-o'-lantern codes

FIGURE 5-16. With a different set of DATA lines, the HALOWEEN program produces this penguin.



produces the penguin drawing in figure 5-16. In this case, using a lot of five-, six-, and seven-dot codes (BS codes of 124, 126, and 127, respectively), we get a figure that has about equal areas of black and white.

By now you should be wondering: how do we come up with the dot codes in the first place? This is the same as asking how we digitize a drawing, that is, how we convert the lines and shapes of the original drawing to a set of numbers representing the printer's different dot patterns. In screen graphics, you can use a tool connected to your computer called a digitizer to generate the screen locations by manually tracing a drawing or doing freehand sketches on the surface of a "touch pad" (e.g., Koala Technologies' Touch Pad or KoalaPad). More advanced devices, such as the ThunderScan digitizer recently introduced for the Apple Macintosh, have eliminated nearly all the manual work, so you can digitize a drawing or photograph in an automatic scanning fashion. And with a screen-dump utility, of course, whatever is created on the computer's screen by these methods can be transferred to a printer.

But let's assume that your system doesn't have either an automatic or semiautomatic digitizer, and you don't want to do printer graphics by screen dump (even with the Macintosh's nifty screen graphics) in the first place. With these conditions you are forced to work out a completely manual method for converting the visual elements of a drawing to a series of dot codes.

For a simple freehand drawing, a good way to start is to make a rough pencil sketch on a piece of graph paper that has eight or ten squares per inch. An even better way is to use the printer to produce long sheets of coding paper having seven-by-seven matrices of open circles. You can see what these look like in figure 5-17 and figure 5-18 and how to make them with program 5-9, GRID7X7. If your printer has a well-

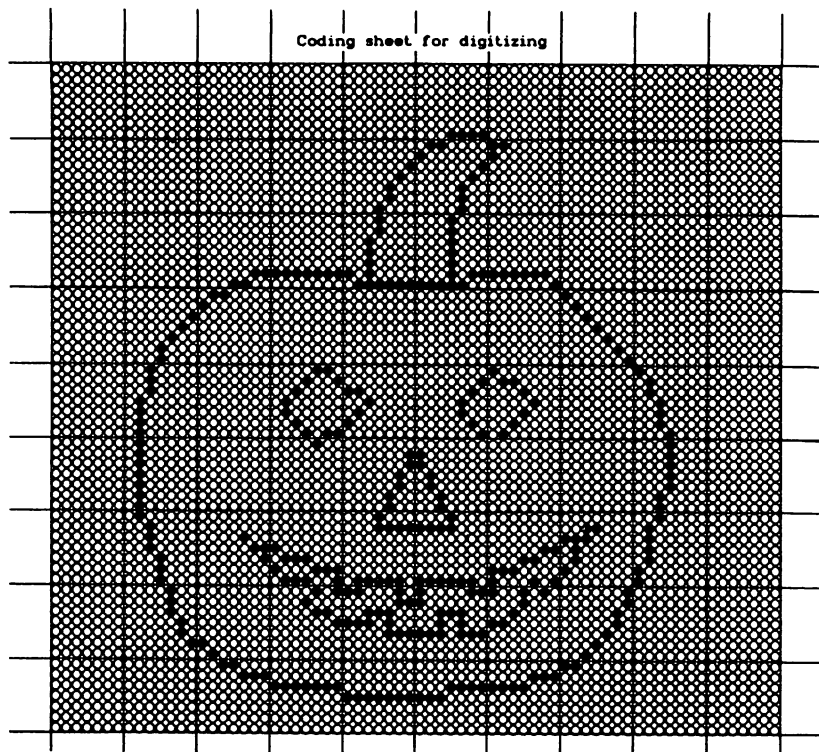


FIGURE 5-17. Coding sheet for digitizing the jack-o'-lantern drawing. The seven-by-seven matrices of open circles are produced by program 5-9, GRID7X7.

PROGRAM 5-9. GRID7X7. Printing a coding sheet for digitizing.

```

10 'PROGRAM 5-9: "GRID7X7" -- FOR MAKING 7-BY-7 CODING SHEETS
15 'FOR TRS-80 WIDEBODY PRINTERS (FOR EIGHT-INCH PRINTERS, CHANGE '18' TO '11'
16 'IN LINE 100 AND SHORTEN LINE 190 TO 770 REPETITIONS OF CODE 129)
20 CLEAR 2000
30 P0$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0) 'CR WITHOUT LF
40 D0$=CHR$(18)+CHR$(156)+CHR$(162)+STRING$(6,193)+CHR$(162)+CHR$(156)
50 INPUT "NUMBER OF 7-DOT ROWS";RW
60 LPRINT CHR$(30);CHR$(27);CHR$(20); 'CONDENSED PRINTING
100 FOR X=1 TO RW
110   FOR L=1 TO 7
120     LPRINT P0$;
130     FOR Y=1 TO 18
140       FOR Z=1 TO 7: LPRINT D0$;: NEXT Z
150       LPRINT CHR$(255); 'VERTICAL LINE SEPARATING MATRIX UNITS
160     NEXT Y
170     LPRINT CHR$(13);
180   NEXT L
190   LPRINT P0$;STRING$(250,129);STRING$(250,129);STRING$(250,129);STRING$(250,
129);STRING$(200,129);STRING$(60,129);CHR$(27);CHR$(91); 'HORIZONTAL LI
NE SEPARATING ROWS
200 NEXT X

```

worn ribbon or one with a light-colored ink, the drawing will stand out more clearly than if the circles are printed in heavy black, and this will make coding easier.

Figure on drawing your original sketch about ten times larger (in height and width) than the final printout; if you want the figure in the printout to come out around two inches high, then make the sketch about twenty inches high. Also, leave an inch or so of blank area around all four sides of the drawing, in case you want to print it in a black-white reversal. If your coding sheet isn't wide enough for the large sketch, glue two or several of the sheets together, as needed.

After you've decided on the final location of every line in the sketch, number the rows and columns of seven-by-seven squares in a fashion that is similar to the numbering of the coding sheet for the penguin drawing. If you plan to print the figure so that its vertical axis is parallel to the edge of the printer's paper, consider the rows to be print lines, each seven dots high. Unless there are too many codes in a print line to fit into a single DATA statement, the numbering of the print lines can also correspond to the numbers used in the program for DATA lines. For that reason, it's a good idea to start the numbering well above the number of the last line of the main program—notice that the numbering for the jack-o'-lantern and penguin starts at 210.

If you want the vertical axis of the figure to be at right angles to the paper's edge, visualize the printhead as sweeping up the columns, instead of across the rows, of the coding sheet, and turn the sheet ninety degrees for the coding. In the case of the jack-o'-lantern we had nine rows of seventy dot columns (ten seven-dot-column groups) each in a "parallel" orientation. If we had used an orientation turned ninety degrees, we would have had ten "rows" of sixty-three dot columns (nine seven-dot-column groups) each, with the pumpkin's head pointing to the right.

Next comes an important step and one that is easy to glide over too hurriedly: filling in the tiny circles of the coding sheet so that each of them represents what will be a printed dot in the final product. Within each seven-by-seven matrix you have to make a firm decision about which dots in each dot column you want to turn on and which you want to remain blank. Often a line of the drawing will run exactly between two of the open circles, and you may feel at a loss to

decide which of the two should become dot locations or whether both should. It may not matter in the long run, because the printout will be so much smaller than the drawing for the coding, but the point is that if you don't decide one way or another, you won't know which dot-code number to assign in the digitizing.

The hardest work, of course, is that digitizing. To make it easier, you should fill in the open circles fully with a dark pencil where dot locations are to be, and unsmudge where you've made erasures or vague lines, so that you have the maximum visual contrast between "dot" circles and "blank" circles. (If you're using graph paper, read "squares" for "circles.")

From that point on, the procedure is simply to go from one dot column to the next, determining the dot code of each. If you've never done this before, you may need to have figures 4-3 and 4-7 to refer to for translating the dot patterns into dot codes, but mainly for psychological comfort. The fastest way, clearly, is to memorize the binary bit values of each dot location in a dot column (i.e., on TRS-80 printers, the sequence 1, 2, 4, 8, 16, 32, and 64 for the top through bottom dot locations, respectively), and get the dot code by adding these bit values in each dot column. For example, if you want the top and bottom pins of the printhead to print dots, then the code is simply the sum of 1 (top pin) and 64 (bottom pin), or 65; if you want the middle three, then it's $4 + 8 + 16$, or 28.

You'll want to do all these additions in your head to save time, and this isn't too difficult after some practice as long as you're dealing with only seven-dot columns that have code numbers from 0 to 127. But if you have eight pins to cope with, the task suddenly becomes twenty times harder and more maddening, because now the range is doubled (0 to 255) and you may often have to deal with adding combinations such as $128 + 64 + 4 + 2 + 1$ or $2 + 8 + 16 + 32 + 128$. This should be fair warning to IBM-Epson and Apple users; they will be wise to stick to seven pins even though their printers have eight that are addressable. There is another advantage to the seven-pin approach, too—free and easy translation of TRS-80 codes to IBM-Epson or Apple codes and vice versa (see programs 4-2 and 4-3).

In digitizing simple drawings such as our jack-o'-lantern and penguin, things usually proceed smoothly if you do one

row at a time, writing down each dot code or each repetition of a dot code (using the negative-number repeating device) on lined notebook paper, and typing all the codes into DATA statements at some later time. If you divide each row into seven-dot-column units in accordance with the straight lines on the coding sheet (these should be inked by hand if you're using graph paper), and check to make sure that you have exactly seven dot codes written down for each of these units before going on to the next, you're much less likely to make coding errors. You'll also be in an easier position to check your written codes against a listing of the codes you typed into the computer.

This practice is illustrated in the coding for the penguin drawing in figure 5-18 (the procedure was not followed in the coding of HALLOWEEN). Notice in the DATA lines of figure 5-18 that, except for the consecutive blanks at the beginning and end of each row, the coding is chopped into seven-code units. In line 230, for example, the first two units having fourteen codes ($-8, 0, 32, -5, 112$) are followed by the third unit ($-6, 112, 120$), the fourth unit ($120, -2, 124, -3, 126, 127$), and the fifth unit ($-7, 127$) of seven codes, even though all five units could be condensed to $-8, 0, 32, -11, 112, -2, 120, -2, 124, -3, 126, -8, 127$. While more laborious, this seven-code-unit method saves a lot of time later in editing, not only by reducing the number of coding errors, but also by making it easier to spot where they have occurred.

If you are digitizing in order to copy a drawing instead of starting with a freehand sketch, the techniques can be much the same, except that you will probably need to start with a tracing that is a lot smaller than the drawing on the coding sheet. If the tracing is done on a piece of see-through graph paper, you can do the necessary enlargement by letting each square of the graph paper be represented by a seven-by-seven square matrix of circles or squares. The penguin, for instance, was constructed from a tracing of a figure that was originally only about two inches high, and the tracing was made on a small piece of graph paper having eight squares to the inch. Then, using the old-fashioned eye-and-hand method, the lines of the tracing were translated to the lines of the enlarged coding sheet in a square-by-square manner, as indicated in figure 5-18. This method can, of course, be

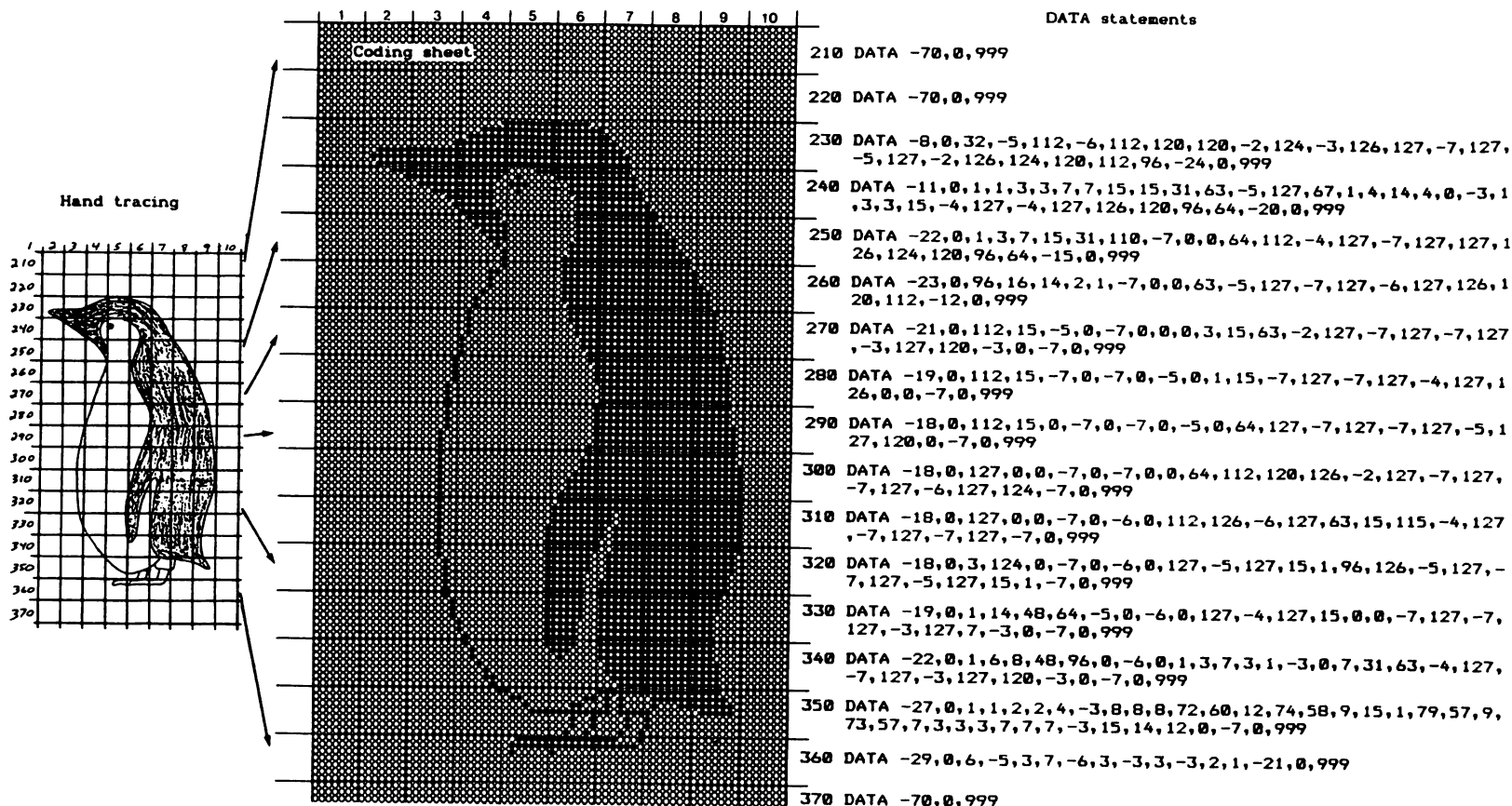


FIGURE 5-18. Original hand tracing (left), coding sheet (center), and DATA statements (right) for the penguin drawing.

used for enlarging a small freehand drawing as well as a tracing.

In using a seven-by-seven matrix unit in our coding sheet, we are assuming that the print density will be set so that the ratio of horizontal to vertical distance in the printout will be one-to-one. This means we should use the elite (72-dots-per-inch) density on TRS-80 printers, and similar densities on IBM-Epson and Apple printers. The penguin printout in figure 5-16 was made with the elite density for the DMP-400, set by `LPRINT CHR$(27);CHR$(23);`. What happens when the same set of dot codes is printed in different densities is shown by figure 5-19.

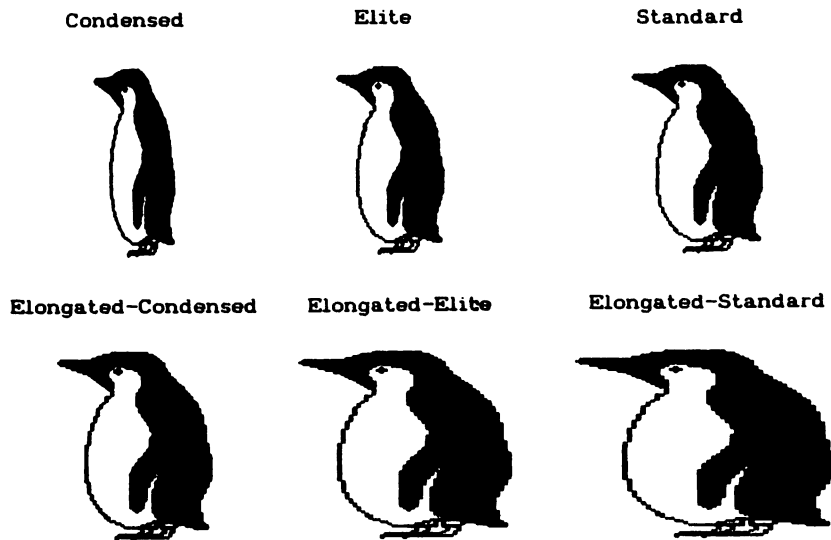


FIGURE 5-19. In digitizing, you need to know which print density will be used in printing the final product. Here, the penguin, designed for the one-to-one elite density, is printed in all the other densities of the DMP-400.

As you can see, you'd better know ahead of time how you intend to print a picture before setting up the coding sheet. If you want to use the DMP-400's standard density for printing (60 dots per inch) and you don't want it to look distorted, then a six-by-seven matrix of circles or squares should be your basic unit in the coding sheet instead of the seven-by-seven matrix we've been using.

How easy are black-white reversals of whole pictures? Just as easy as with the little black arrow back in chapter 4 (figures 4-4 and 4-5). Simply change `LPRINT CHR$(128+N);` to `LPRINT CHR$(255-N);` in the program line that prints single dot patterns and `LPRINT STRING$(-N,128+M);` to `LPRINT STRING$(-N,255-M);` in the line that repeats a dot pattern. Figure 5-20 shows how the penguin and some more products of HALLOWEEN-type programs (called DAGWOOD and RABBITS) are transformed in this fashion.

For larger and more complex line drawings, we need to refine our procedures in several ways. First of all, we'll get better pictures, because of higher resolution, if we use a more condensed density than either of these. For the DMP-400's highest density (100 dots per inch) or the Imagewriter's "Elite" density (96 dots per inch), ten-by-seven matrix units work well, and the IBM-Epson double density (120 dots per inch) calls for a twelve-by-seven size. For the Imagewriter's highest density (160 dots per inch), we should increase that to about eighteen-by-seven, and for the Epson FX-80's quadruple density (240 dots per inch) to twenty-four-by-seven.

If we use the ten-by-seven ratio for the DMP-400, we'll need a coding sheet or graph paper having vertical lines every ten dot columns instead of every seven. Three small changes in GRID7X7 (program 5-9) will convert it to GRID7X10—with widebody printers, change 18 to 13 in line 130, 7 to 10 in line 140, and 60 to 100 in line 190.

Unless we compress the circles or squares so that ten of them take no more horizontal space than seven did before (which would make it a lot harder to distinguish dot columns), we will have to make our enlarged drawing from a tracing of a distorted one—about 40% fatter than normal. This is not too difficult to do if, as before, you make the enlargement in a square-to-square (now square-to-rectangle) fashion.

Figure 5-21 shows how you can trace a photograph of your favorite cat and blow up the tracing for a ten-by-seven coding sheet. In this case an eight-by-ten-inch photograph was traced, and the grid of the tracing paper was only four squares per square inch, so the degree of enlargement vertically was only about 25%. Horizontally, it had to be doubled to fit the ten-by-seven coding format.

The digitizing was done as before, except that groups of

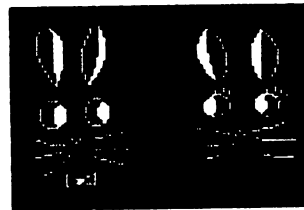
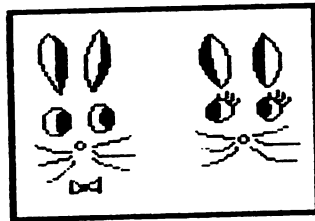


FIGURE 5-20. Once you work out the dot codes, you can manipulate them in various ways. The easiest way is illustrated by these black-white reversals. (Dagwood reproduced with special permission of King Features Syndicate, Inc.)

ten rather than seven dot columns were coded in each line. After all fifteen rows of 180 codes each were digitized in handwritten form, other refinements were introduced in the arrangement of DATA statements (see program 5-12 on pages 154–155). The whole drawing was divided horizontally into three segments of sixty dot columns each, to prepare for en-

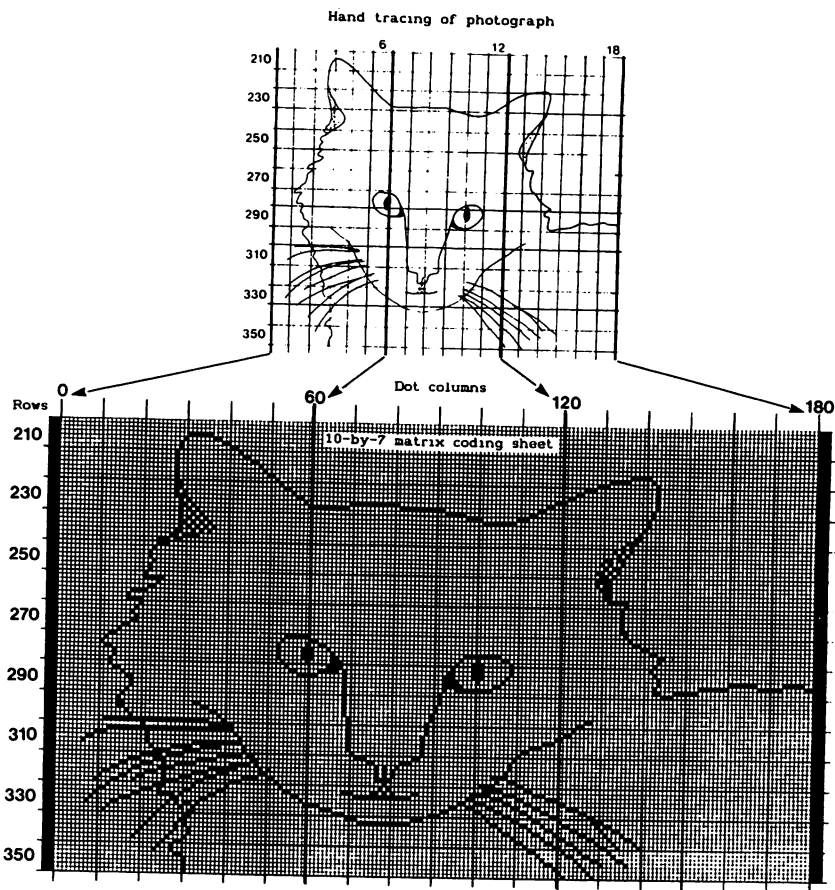


FIGURE 5-21. Digitizing for the DMP-400's condensed (100-dots-per-inch) density. In working out the codes for this tracing of a photograph (upper part of figure), each square of the tracing is translated to a ten-by-seven rectangle of the coding sheet (lower part of figure), resulting in an enlargement that is distorted.

tering, printing, and editing the dot codes for one third of the drawing at a time.

For the left third of the figure, automatic line numbering of the DATA statements was started with the command **AUTO 210**, and the first sixty dot codes of each print line were entered. Thus DATA lines 210, 220, 230, and so on were used for the first sixty codes of the first, second, third, etc., print lines, respectively. As these were typed in, they were placed after DATA -3, 127, so that every print line would start with a solid vertical line; this served as a check on carriage returns. Then, after the command **AUTO 219**, the end-of-line codes were entered in the form **DATA 999** in lines 219, 229, 239, and so forth.

Next, the left third of the drawing was printed, using a

main program that was an enhanced version of HALOWEEN called WHITECAT. One of its enhancements was the addition of a byte counter that would count every dot pattern printed in a print line. Another was changing the FOR R = ... statement in line 100 so that R (row) values would conform to the numbering of DATA and print lines. Then, just after the end of each print line (or partial print line, before the drawing was completed), a print of the line number and the number of codes printed was introduced, so that editing would be even easier to do.

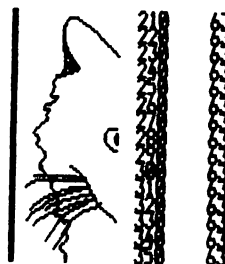
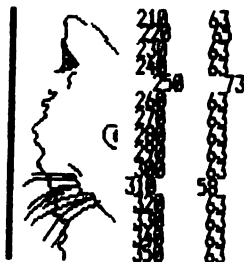
The printouts in figure 5-22 show how the construction and editing of the picture proceeded as the middle and right-hand segments were added. In the upper left part of the figure is the printout after the first third of the codes were entered. You can see in this example how clearly coding errors showed up, how easy it was to identify the lines in which they occurred, and how many missing or extra dot codes they involved. After all the errors were corrected, the upper right printout verified that all lines had the same number of bytes, so that work on the middle third could start. The dot codes for that segment were entered (using **AUTO 213** and lines 213, 223, 233, . . . , 353). These were displayed in a printout to find coding errors, and the DATA lines were cleansed of those errors before proceeding to the final segment (see the two middle figures). When the final set of dot codes was entered (in lines 216, 226, 236, . . . , 356), three codes for another vertical line (–3, 127) were added to provide another check on the byte count in each print line. The lower right printout in figure 5-22 shows the final product after the third stage of editing.

The digitizing labor may seem more worth the effort if you consider an original set of codes to be merely a foundation on which to build more creations. Figure 5-23 shows several ways we can use or modify a set of dot codes to produce distinctly different printouts. Notice in the upper left printout in this figure that our WHITECAT codes produced a white cat with black whiskers and their reversal (upper middle printout) produced a black cat with white whiskers. Since white cats and many black cats in the real world have white whiskers, the reversed version of the cat came out looking more like a real cat than the original white cat did. But the reversal of the eyes and the blackening of the background

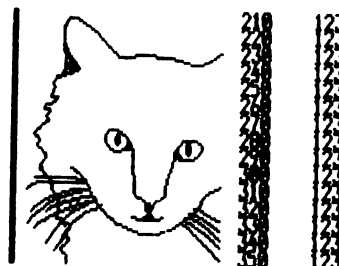
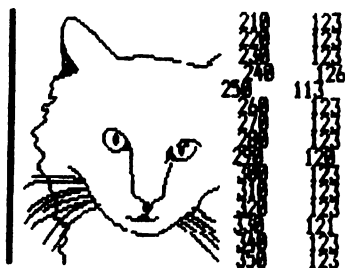
Before editing:

After editing:

First segment



Middle segment added



All three segments

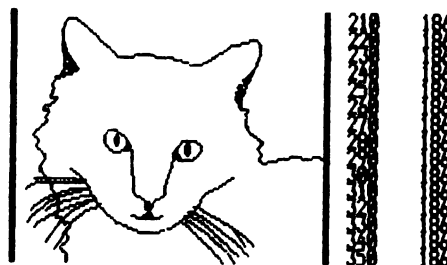
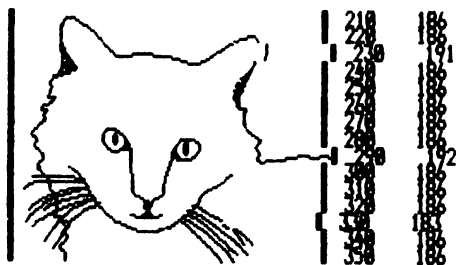


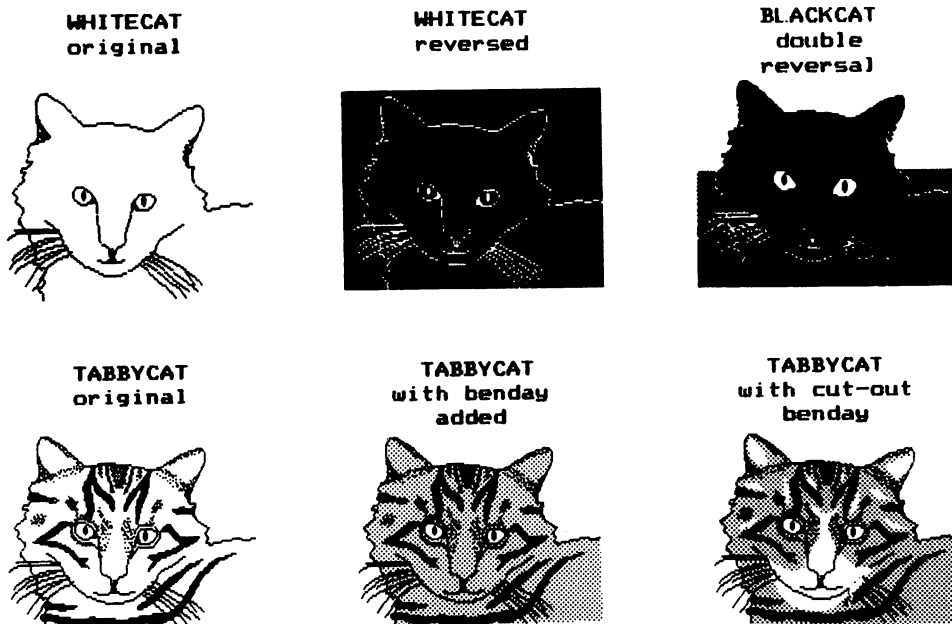
FIGURE 5-22. When the number of dot codes in each print line is over a hundred, spotting errors in coding and correcting them is much easier if you do the drawing in left-to-right segments. Here the codes for WHITECAT are entered and edited in left, center, and right thirds of sixty codes each in 180-code print lines. Three codes are added for vertical lines at the beginning and end of each print line for checking carriage returns and line lengths.

area around the cat still rendered the reversed version pretty unrealistic.

Another program, called **BLACKCAT**, remedies these shortcomings mainly by changing some of the 0 codes that in the reversal come out black to 127 codes that print blank, and vice versa. This double reversal was applied to the upper part of the background and the cat's eyes; some of the background was changed to dark shading (by alternating dot codes 42 and 85 or repeating the series 73, 18, and 36) so that the white whiskers would still show up against a dark background (upper right printout). No more than a quarter of the original **WHITECAT**'s dot codes were changed, some by re-typing entire **DATA** lines and some by merely inserting short sequences of codes in the original lines. The dot codes were subtracted from 255 instead of being raised by 128 so that unchanged codes would continue to print in black-white reversal.

Three more printouts in figure 5-23, displayed in the bottom row, were produced by returning to the **WHITECAT** coding sheet and adding stripes and other markings. The codes in the **WHITECAT** program were changed accordingly to create a new program called **TABBYCAT**.

FIGURE 5-23. A variety of printouts from the **WHITECAT** program and its modified versions, **BLACKCAT** and **TABBYCAT**. Screen tints ("bendays") were added to the last two printouts (bottom row).



The bottom left printout is from **TABBYCAT**, as produced on the printer without any manual finishing touches. Its main purpose is to show how much more detailed a bit-image drawing can be when you use a print density that is higher than the one used in the jack-o'-lantern and penguin drawings. The center panel shows what happens when you apply one of the favorite tricks of cartoonists—adding a screen tint (also called a “benday”) to produce a medium gray. After all, most tabby cats are gray with black markings. In the right panel we have an even prettier tabby that is gray with black and white markings, in which parts of the gray benday material have been cut out with an artist's knife to expose the white paper underneath. We'll soon see other ways of getting grays without adding these plastic tints by hand.

All of these recent programs—**HALOWEEN**, **PENGUIN**, **DAGWOOD**, **RABBITS**, **WHITECAT**, **BLACKCAT**, and **TABBYCAT**—are merely the offspring of a generic line-drawing program for TRS-80 printers called **LINEDRAW**. Three different versions of this program are presented in programs 5-10, 5-11, and 5-12, for Tandy, IBM-Epson, and Apple printers, respectively. These programs contain all the enhancements that we noted in the transition from **HALOWEEN** to **WHITECAT**, and some additional features.

They also contain dot codes for three different drawings which are all designed for TRS-80 printers but will run on IBM-Epson and Apple printers as well. **LINEDRAW/TRS** has the DATA lines from **DAGWOOD**, which was digitized for 100-dots-per-inch printing. At the nearest lower density of the FX-80 (90 dots per inch) the Dagwood figure will be somewhat fatter than normal, and at the next higher density (120 dots per inch) it will be too thin; with Apple printers there will be little discrepancy when you use 96 dots per inch. **LINEDRAW/EPS** has dot codes of another cartoon from a program called **WHITEDOG**. These codes were programmed for 60 dots per inch on TRS-80 printers and are followed by a code-conversion subroutine in **LINEDRAW/EPS** that can be used with any seven-bit TRS-80 or Apple codes; the subroutine is a slightly modified version of **TRSTOEPS** (program 4-3). If you need to convert eight-bit Apple codes to IBM-Epson codes, use the **EPSTOAPL** subroutine (program 4-2) in reverse. **LINEDRAW/APL** presents

the data codes from the WHITECAT program, designed for 100-dots-per-inch density.

At this point our LINEDRAW programs are only a little more sophisticated than the similar programs in the Epson and TRS-80 printer manuals. But if we take things further, we may get closer to more "photographic" drawings.

PROGRAM 5-10. LINEDRAW/TRS. Universal line-drawing program for TRS-80 printers.

```

10 'PROGRAM 5-10: "LINEDRAW/TRS" -- UNIVERSAL LINE DRAWING PROGRAM
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS (USE BOOTHE'S PRINTER DRIVER IF MOD. III)
30 CLEAR 1000                                '(NOT NECESSARY FOR MODEL 4)
40 DEFINT C,M,N,R,Z                          '(TO SPEED UP PRINTING)
50 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
60 LPRINT CHR$(30);CHR$(27);CHR$(20);        'SET CONDENSED PRINT DENSITY
70 LPRINT CHR$(18);                          'ENTER GRAPHIC MODE
100 FOR R=210 TO 450 STEP 10: C=0             'LOOP FOR 25 ROWS, RESET COUNTER
110   READ N: IF N=999 THEN GOTO 170           'READ CODE #, LINE FEED IF 999
120   IF BW$="B" AND N>=0 AND N<128 THEN LPRINT CHR$(128+N);: C=C+1: GOTO 110
      'PRINT DOT-PATTERN AND ADVANCE BYTE COUNTER
130   IF N>=0 AND N<128 THEN LPRINT CHR$(255-N);: C=C+1: GOTO 110
      'PRINT IN BLACK-WHITE REVERSAL AND ADVANCE BYTE COUNTER
140   READ M                                   'CODE # WAS NOT POSITIVE OR ZERO, SO READ NEXT
150   IF BW$="B" THEN LPRINT STRING$(-N,128+M);: C=C+ABS(N): GOTO 110
      'REPEAT-PRINT DOT-PATTERN AND ADVANCE BYTE COUNTER
160   LPRINT STRING$(-N,255-M);: C=C+ABS(N): GOTO 110
      'REPEAT-PRINT WHITE-ON-BLACK AND ADVANCE BYTE COUNTER
170   PRINT "BYTE COUNT FOR ROW";R;" =";C: LPRINT STRING$(5,128);CHR$(30);R;" "
      ;C;CHR$(18);
180   FOR Z=1 TO 7: LPRINT CHR$(27);CHR$(50);: NEXT Z: LPRINT CHR$(27);CHR$(16);
      CHR$(0);CHR$(0);
190 NEXT R
195 END
200 REM -- DOT CODES FOR "DAGWOOD" CARTOON DRAWING
210 DATA -142,112,999
220 DATA -3,127,-136,0,-3,127,999
230 DATA -3,127,-86,0,127,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
240 DATA -3,127,-86,0,127,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
250 DATA -3,127,-48,0,16,8,16,16,36,68,72,8,48,64,-6,0,-5,64,-6,0,-3,64,-4,0,64,
64,32,32,88,80,84,40,56,16,16,112,-6,0,127,-14,0,127,-12,0,127,-7,0,-3,127,9
99
260 DATA -3,127,-47,0,2,2,4,4,4,5,5,9,9,10,20,121,126,38,3,115,11,11,75,91,3,3,7
,86,75,11,19,103,7,15,15,7,51,107,89,84,106,42,119,77,12,10,10,5,5,7,6,6,-4,
2,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
270 DATA -3,127,-55,0,112,8,4,99,47,113,96,17,18,16,-2,103,99,0,0,7,7,3,0,1,0,0,
8,18,23,79,72,15,31,24,111,3,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
280 DATA -3,127,-55,0,3,4,4,14,57,64,1,-5,2,1,32,32,96,64,-4,0,16,56,51,116,52,2
0,19,8,4,3,-14,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999

```

```

290 DATA -3,127,-52,0,64,64,32,32,16,16,80,62,2,3,6,68,72,80,80,112,-5,48,112,48
,24,8,12,6,5,2,2,114,44,32,64,-14,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
300 DATA -3,127,-48,0,112,8,6,1,-3,0,96,16,0,7,-5,8,-3,4,7,-4,4,-3,3,5,5,8,16,16
,76,35,16,0,0,0,1,1,2,2,4,56,64,-7,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
310 DATA -3,127,-46,0,120,7,-5,0,64,60,3,-20,0,112,12,2,1,64,56,-8,0,64,60,3,-7,
0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
320 DATA -3,127,-43,0,96,28,3,-6,0,112,15,0,0,-4,8,126,-4,1,97,30,16,16,-7,0,7,8
,80,56,7,-9,0,127,-9,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
330 DATA -3,127,-40,0,112,12,3,-4,0,64,32,32,48,46,33,-8,0,1,2,2,1,-8,0,64,32,24
,4,2,1,-10,0,126,1,-9,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
340 DATA -3,127,-39,0,3,68,56,-4,64,3,4,4,68,60,4,4,116,12,4,-4,36,42,42,116,12,
100,12,8,8,16,16,56,100,34,33,-7,64,56,-5,0,112,14,1,-10,0,127,-14,0,127,-12
,0,127,-7,0,-3,127,999
350 DATA -3,127,-39,0,120,6,1,17,97,1,11,82,99,119,-14,127,126,127,-2,126,124,60
,60,23,-3,16,48,32,96,96,99,98,97,64,126,2,2,2,1,-13,0,127,-14,0,127,-12,0,1
27,-7,0,-3,127,999
360 DATA -3,127,-39,0,3,76,-2,112,121,126,-17,127,3,1,113,1,0,120,0,0,124,0,0,96
,-2,112,48,24,12,3,1,1,-4,0,127,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,99
9
370 DATA -3,127,-33,0,64,96,112,120,124,126,-23,127,126,112,97,92,112,64,7,24,96
,67,76,56,11,4,-10,0,127,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
380 DATA -3,127,-28,0,96,120,124,126,-37,127,7,1,1,-14,0,127,-13,0,127,-14,0,127
,-12,0,127,-7,0,-3,127,999
390 DATA -3,127,-3,1,-6,2,-6,4,-6,8,-6,16,-32,127,-3,63,55,55,104,83,83,94,97,-1
7,0,127,-13,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
400 DATA -3,127,-3,2,-6,4,-6,8,-6,16,-6,32,47,-41,127,127,124,-3,32,-6,64,-3,0,-
3,64,63,32,-12,0,127,-14,0,127,-12,0,127,-7,0,-3,127,999
410 DATA -3,127,-16,0,64,48,88,40,84,44,84,44,84,44,84,43,87,47,95,63,-37,
127,32,32,-6,64,0,1,1,3,-6,2,-8,4,2,14,18,18,-3,16,-7,32,64,64,32,32,33,-5,3
2,64,-6,0,31,16,16,80,32,-3,0,-3,127,999
420 DATA -3,127,-16,0,15,29,122,117,122,117,122,117,122,117,122,117,122,117,122,
117,122,117,90,53,90,45,91,45,87,45,87,45,87,45,87,45,87,42,87,47,95,63,95,6
3,127,63,127,63,127,63,-7,127,63,103,32,96,64,-5,0,-6,1,-6,2,-6,4,-6,8,-6,16
,-7,32
430 DATA -6,64,1,1,-6,2,1,-5,0,-3,127,999
440 DATA -3,127,-33,0,31,58,117,106,117,106,117,106,117,106,117,106,117,106,117,
106,117,106,117,106,117,106,117,106,117,106,117,106,53,58,53,122,117,122,117
,122,117,122,117,122,117,15,-47,0,-7,1,-7,2,-3,127,999
450 DATA -3,127,-38,0,-15,1,-83,0,-3,127,999
460 DATA -142,7,999
470 STOP

```

PROGRAM 5-11. LINEDRAW/EPS. Universal line-drawing program for IBM-Epson printers.

```

10 'PROGRAM 5-11: "LINEDRAW/EPS" -- LINE DRAWING FOR IBM-EPSON PRINTERS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR MICROSOFT-BASIC COMPUTERS (TRS-80 MODEL III: CHANGE ^ TO LEFT BRACKET)
25 'CONVERTS TRS-80 CODES TO EPSON CODES. IF EPSON 8-BIT CODES ARE USED, DELETE
    LINES 135 AND 155. IF APPLE 8-BIT CODES ARE USED, REPLACE SUBROUTINE 5000 W
    ITH MODIFIED "EPSTOAPL" FOR APPLE-TO-EPSON CONVERSION.

```

```

28 'IF EPSON FX-80 IS USED, PRINT DENSITY OF 576 DPI MATCHES TRS-80'S ELITE DENS
    ITY AND APPLE IMAGEWRITER'S LOWEST DENSITY.
30 D$="K": ROWS=11: COLS=56          'D$=PRINT DENSITY
40 LPRINT CHR$(27)"0";                'MASTER RESET
50 LPRINT CHR$(27)"1";                'SET 7/72" LINE FEED
60 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
100 FOR R=1 TO ROWS
105   C=0
110   LPRINT CHR$(27);D$;CHR$(COLS)CHR$(0);    'GRAPHIC RESERVATION
120   READ N: IF N<0 THEN GOTO 150
130   IF N=999 THEN GOTO 170
135   NTRS=N: GOSUB 5000: N=EC          'CONVERSION OF TRS-80 OR APPLE TO EPSON
140   IF BW$="B" AND N>=0 AND N<256 THEN LPRINT CHR$(N);: C=C+1: GOTO 120    'PR
      INT N
145   LPRINT CHR$(255-N);: C=C+1: GOTO 120    'PRINT B-W REVERSAL OF N
150   READ M          'CODE (N) WAS NEGATIVE, SO READ NEXT CODE AND CALL IT M
155   NTRS=M: GOSUB 5000: M=EC
160   IF BW$="B" THEN LPRINT STRING$(-N,M);: C=C+ABS(N): GOTO 120
      'PRINT M, N TIMES
165   LPRINT STRING$(-N,255-M);: C=C+ABS(N): GOTO 120    'B-W REVERSAL OF M
170   PRINT "BYTE COUNT FOR ROW";R;: -";C: LPRINT " ";R;: " ";C;
175   LPRINT CHR$(13);
180 NEXT R
190 END
300 REM -- DOT CODES ARE FOR "WHITEDOG" CARTOON DRAWING
310 DATA -11,0,64,32,-3,16,-4,8,16,16,32,112,72,68,-2,64,-28,0,999
320 DATA -7,0,96,28,2,1,-12,0,1,-4,0,65,33,17,9,-8,1,-3,2,4,8,16,96,-9,0,999
330 DATA -5,0,64,62,3,0,32,64,-16,0,1,-18,0,1,14,116,68,100,72,112,0,0,0,999
340 DATA -3,0,96,28,99,124,126,127,126,124,113,98,14,112,-27,0,64,32,32,16,12,6,
    -4,3,1,0,0,0,999
350 DATA -3,0,15,48,79,63,-5,127,63,28,99,30,-3,32,64,64,-8,0,96,16,16,8,8,-3,4,
    -3,2,1,1,-14,0,999
360 DATA -5,0,1,2,-4,5,4,4,2,1,-6,0,1,1,2,4,120,-3,8,4,5,6,8,48,64,-21,0,999
370 DATA -16,0,8,48,64,-3,0,112,12,3,-3,0,32,64,-4,0,1,2,12,112,-18,0,999
380 DATA -13,0,112,12,4,8,48,67,61,70,127,0,127,0,64,-2,0,64,7,120,-4,0,96,28,3,
    -18,0,999
390 DATA -13,0,3,124,-3,0,1,2,5,7,0,1,17,25,43,69,65,1,120,-3,0,15,8,8,-3,4,2,34
    ,66,2,66,60,-10,0,999
400 DATA -15,0,3,4,8,16,32,64,32,16,64,32,64,64,32,113,10,7,-5,0,64,32,17,18,12,
    4,2,1,-12,0,999
410 DATA -21,0,-4,1,0,0,3,4,4,-3,2,-3,1,-20,0,999
420 STOP
5000 'SUBROUTINE FOR CONVERTING TRS-80 TO IBM-EPSON CODES ("TRSTOEPS")
5020 PRINT "ORIGINAL DECIMAL:";NTRS
5030 FOR P=6 TO 0 STEP -1              '(7 PINS, NUMBERED 6 TO 0)
5040   X(P)=NTRS-2^P                  '(CONVERSION OF
5050   IF X(P)<0 THEN Y$(P)="0": GOTO 5070    'DECIMAL CODE
5060   Y$(P)="1": NTRS=X(P)              'TO BINARY FORM)
5070 NEXT P
5080 BYTE$=Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0)    'ORIGINAL BYTE (BINARY)
5090 PRINT "ORIGINAL BYTE$:";BYTE$
5100 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6)    'TURN BYTE UPSIDE DOWN
5110 PRINT "MIRROR-IMAGE BYTE$:";MBYTE$

```

```

5120 EC=64*VAL(Y$(0))+32*VAL(Y$(1))+16*VAL(Y$(2))+8*VAL(Y$(3))+4*VAL(Y$(4))+2*VAL(Y$(5))+VAL(Y$(6))
      'CONVERT MIRROR BYTE TO DECIMAL
5130 PRINT "ORIGINAL TRS CODE, ";NTRS;" , EQUALS EPSON CODE ";EC
5135 IF EC=9 THEN EC=8
5140 RETURN

```

PROGRAM 5-12. LINEDRAW/APL. Universal line-drawing program for Apple printers.

```

10 REM: PROGRAM 5-12: "LINEDRAW/APL" -- UNIVERSAL LINE DRAWING PROGRAM
15 REM: Copyright (c) 1985 by John Warner Davenport
18 REM: FOR APPLESOFT BASIC AND APPLE DOT MATRIX OR IMAGEWRITER PRINTERS
20 PR#1
30 NL=15: COLS=186
40 PRINT CHR$(27)"E";: REM -- 96 DPI PRINT DENSITY
100 FOR R=1 TO NL
110   READ N: IF N<0 THEN GOTO 150
120   IF N=999 THEN GOTO 180
130   IF PEEK(49305) <> 16 THEN GOTO 130
140   IF N>=0 AND N<256 THEN PRINT CHR$(27)"G0001";: POKE 49304,N: GOTO 110
150   READ M
160   IF PEEK(49305) <> 16 THEN GOTO 160
170   FOR X=1 TO -N: PRINT CHR$(27)"G0001";: POKE 49304,M: NEXT X: GOTO 110
180   PRINT CHR$(27)"T14"; CHR$(13);: REM -- 7/72" LINE FEED, CARR. RETURN
190 NEXT R
195 PR#0
198 END
200 REM -- DATA CODES BELOW ARE FOR "WHITECAT" DRAWING
210 DATA -3,127,-28,0,96,16,-6,8,16,16,32,32,64,64,-18,0
213 DATA -60,0
216 DATA -60,0,-3,127
219 DATA 999
220 DATA -3,127,-27,0,127,0,0,-10,0,0,0,1,2,2,4,4,8,16,32,32,64,-8,0
223 DATA -60,0
226 DATA -8,0,64,64,64,-5,32,-4,16,16,32,64,-7,0,-30,0,-3,127
229 DATA 999
230 DATA -3,127,-27,0,15,123,44,80,32,64,-7,0,-10,0,0,0,1,2,2,4,8,8,16,32
233 DATA -9,32,16,-10,16,16,-9,32,32,32,-5,64,-3,0,-10,0,0,0,64,64,32,32,16,16,8,8
236 DATA 8,4,4,2,2,1,1,1,0,0,-10,0,0,96,31,-7,0,-30,0,-3,127
239 DATA 999
240 DATA -3,127,-23,0,-3,64,32,112,95,36,42,53,42,21,10,20,8,-3,0,-20,0
243 DATA -37,0,-3,1,1,-8,2,1,1,1,-8,0
246 DATA -10,0,-4,0,64,32,96,48,24,4,3,-9,0,-30,0,-3,127
249 DATA 999
250 DATA -3,127,-21,0,108,18,1,1,1,-4,0,-30,0
253 DATA -60,0
256 DATA -8,0,32,104,84,44,81,54,13,2,1,0,0,0,-40,0,-3,127
259 DATA 999
260 DATA -3,127,-16,0,96,-3,80,19,10,10,6,2,-5,0,-30,0
263 DATA -60,0
266 DATA -8,0,3,5,31,63,-3,32,96,-4,0,-40,0,-3,127
269 DATA 999

```

```

270 DATA -3,127,-14,0,64,32,32,20,26,1,-40,0
273 DATA -60,0
276 DATA -14,0,63,96,-4,0,-40,0,-3,127
279 DATA 999
280 DATA -3,127,-10,0,4,10,73,49,-6,0,-32,0,60,66,-4,1,57,125
283 DATA 58,2,4,4,8,112,96,64,0,0,-25,0,64,32,16,16,80,112,80,-4,16,32,32,64,0,-
10,0
286 DATA -16,0,1,2,2,4,4,8,72,48,16,-5,0,-30,0,-3,127
289 DATA 999
290 DATA -3,127,-10,0,0,0,1,2,2,4,72,56,0,0,-30,0,-4,0,1,2,2,4,4,4
293 DATA 4,4,2,2,-3,3,1,126,0,-20,0,0,64,60,30,31,56,48,32,32,39,47,39,32,32,16,
16,8,4,3,0,-10,0
296 DATA -20,0,0,1,50,44,-3,64,-3,32,-3,32,-7,16,-7,8,-3,16,-3,16,-5,8,-2,16,-3,
127
299 DATA 999
300 DATA -3,127,-10,0,0,-3,64,78,77,80,96,64,64,-10,64,-2,64,68,68,72,72,16,16,3
2,64,-20,0
303 DATA -8,0,31,96,-10,0,-8,0,64,48,12,3,-8,0,-20,0
306 DATA -4,0,-2,64,-2,32,0,0,-50,0,-3,127
309 DATA 999
310 DATA -3,127,-6,0,16,8,8,4,4,4,-8,2,3,30,34,-4,66,-3,34,98,98,82,82,50,50,-3,
51,19,19,19,20,24,96,-5,0,-10,0
313 DATA -9,0,63,64,-9,0,-7,0,124,3,0,-20,0,64,32,32,-3,16,-3,8,4
316 DATA 4,2,2,1,-6,0,-50,0,-3,127
319 DATA 999
320 DATA -3,127,-9,0,64,32,16,16,72,72,36,36,18,74,42,37,21,21,19,11,126,73,73,6
9,37,36,36,20,82,82,50,42,42,25,25,25,77,77,69,36,37,35,38,24,16,40,32,64,-7
,0
323 DATA -10,0,0,-3,1,2,2,60,32,124,32,24,4,-3,2,1,1,-3,0,-10,0,0,-3,64,80,48,16
,24,38,33,64,64,-8,0
326 DATA -60,0,-3,127
329 DATA 999
330 DATA -3,127,-7,0,16,8,4,2,66,33,16,16,-3,8,4,4,-3,2,1,71,37,20,12,24,36,18,8
2,97,16,8,8,4,4,2,2,1,-9,0,-3,0,1,2,2,4,4,8,16
333 DATA -3,16,-3,32,64,64,66,2,2,-4,4,6,6,5,6,5,6,-4,4,2,2,0,0,64,64,64,32,32,1
6,16,8,8,20,22,42,42,75,19,21,37,37,74,74,18,20,36,41,73,18,18,34,36,68,68
336 DATA 8,8,-3,16,32,32,64,64,0,-50,0,-3,127
339 DATA 999
340 DATA -3,127,-10,0,-8,0,32,16,8,4,2,1,0,64,32,16,76,66,39,25,-8,0,-20,0
343 DATA -9,0,1,1,1,-8,2,-5,2,-3,1,0,0,-10,0,-3,0,1,2,2,4,8,8,17,33,34,66,4,9,9,
18,36,36,72
346 DATA 81,18,34,36,68,8,8,16,17,35,69,74,18,36,68,8,16,16,32,64,-40,0,-3,127
349 DATA 999
350 DATA -3,127,-23,0,2,1,0,0,3,4,4,120,-9,0,-20,0
353 DATA -53,0,1,1,2,4,8,8,16
356 DATA 32,65,2,2,4,9,18,34,68,4,8,0,1,2,4,4,8,-3,0,-40,0,-3,127
359 DATA 999

```

Textured drawings. Code 999, for ending a print line, is only one of several special codes a LINEDRAW program might have. For more ambitious picture-drawing, other code

numbers above 255 and below 999 could be used to branch to different subroutines. These could be designed to add shadings, backspace and overprint certain lines, etc., and could include some more timesaving devices.

Instead of a black frame like the one surrounding Dennis the Menace (see figure 1-2), we could add shading for spotlighting, as in figure 5-24.

Random selections of dot codes fed into a dot-matrix printer will give you a nice selection of shadings to use in pictures. The computer's RND (random) function (which is not truly random in its selections) can be used to make selections of codes from groupings of dot codes that print many dots or very few dots, or any dot density from pure white to pure black. To add the shading around Dennis, six new DATA lines were added to LINEDRAW/TRS, and the start of the DATA statements was shifted from line 200 to line 5000. The following lines were added to the main program.

```
125 IF N>500 AND N<511 THEN GOSUB 1000: GOTO
    110
1000 'MULTI-BRANCH FOR FIXED-LENGTH SHADINGS
1010 RANDOM: ON N-500 GOSUB 801,802,803,804,
    805,806,807,808,809,810
1020 RETURN
```

Subroutine 1000 is something of a Pandora's box—an entry to 110 other subroutines. Ten of these, numbered 801–810,

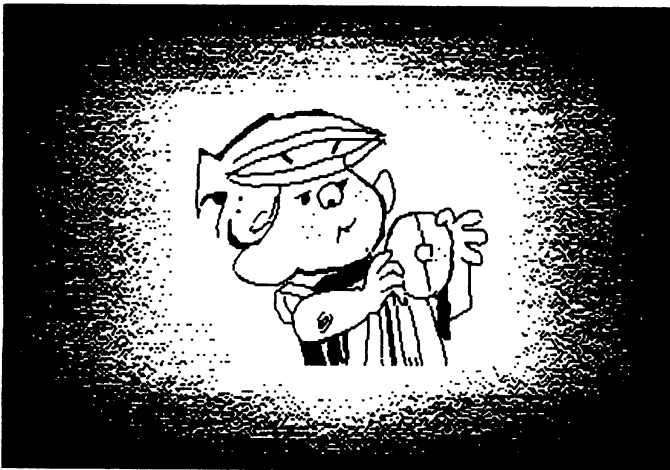


FIGURE 5-24. Spotlighting by adding various densities of shading that are produced by random selections from restricted groups of dot codes. Ten shading levels are generated by a subroutine (program 5-13, SPOTLITE) that is appended to the LINEDRAW/TRS program.

use the RND function to make random selections of dot codes listed in 100 others, numbered 601–700. For each of ten degrees of shading ranging from $\frac{2}{70}$ (3%) to $\frac{63}{70}$ (90%), there are restricted groups of ten dot codes in subroutines 601–700 that may be randomly selected. Any DATA value from 501 through 510 will cause both a jump to subroutine 1000 and a sequence of ten dot prints; the higher the number is in that 501–510 range, the darker the shading will be.

All eleven subroutines that do the selecting of shadings use BASIC's powerful ON...GOSUB switch. The selecting subroutines and groups of dot codes defining the ten degrees of shading are presented in program 5-13, called SPOTLITE. The labels for the latter ($\frac{2}{70}$, $\frac{7}{70}$, etc.) refer to the average number of dots printed at random in a group of ten successive prints, the maximum possible being 70 (all seven dots in all ten prints.)

PROGRAM 5-13. SPOTLITE. Varying degrees of shading for spotlighting a drawing.

```

10 'PROGRAM 5-13: "SPOTLITE" -- SUBROUTINES FOR SHADING LEVELS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 MODEL III AND TRS-80 PRINTERS
30 'NOTE: SHADING LEVELS ARE CALLED IN THE MAIN PROGRAM BY N VALUES OF 501-510
40 'AND THE SELECTIONS ARE ROUTED THROUGH SUBROUTINES 1000 AND 801-810
600 '=====CODES FOR RANDOM SELECTIONS=====
601 OUT 251,128: RETURN          '2/70
602 OUT 251,144: RETURN
603 OUT 251,128: RETURN
604 OUT 251,128: RETURN
605 OUT 251,128: RETURN
606 OUT 251,128: RETURN
607 OUT 251,128: RETURN
608 OUT 251,130: RETURN
609 OUT 251,128: RETURN
610 OUT 251,128: RETURN
611 OUT 251,128: RETURN          '7/70
612 OUT 251,132: RETURN
613 OUT 251,160: RETURN
614 OUT 251,128: RETURN
615 OUT 251,192: RETURN
616 OUT 251,129: RETURN
617 OUT 251,130: RETURN
618 OUT 251,128: RETURN
619 OUT 251,136: RETURN

```

```
620 OUT 251,129: RETURN
621 OUT 251,129: RETURN      '14/70
622 OUT 251,130: RETURN
623 OUT 251,132: RETURN
624 OUT 251,136: RETURN
625 OUT 251,144: RETURN
626 OUT 251,160: RETURN
627 OUT 251,192: RETURN
628 OUT 251,162: RETURN
629 OUT 251,148: RETURN
630 OUT 251,193: RETURN
631 OUT 251,201: RETURN      '21/70
632 OUT 251,162: RETURN
633 OUT 251,166: RETURN
634 OUT 251,148: RETURN
635 OUT 251,193: RETURN
636 OUT 251,146: RETURN
637 OUT 251,137: RETURN
638 OUT 251,148: RETURN
639 OUT 251,161: RETURN
640 OUT 251,138: RETURN
641 OUT 251,170: RETURN      '28/70
642 OUT 251,212: RETURN
643 OUT 251,180: RETURN
644 OUT 251,150: RETURN
645 OUT 251,201: RETURN
646 OUT 251,149: RETURN
647 OUT 251,197: RETURN
648 OUT 251,178: RETURN
649 OUT 251,162: RETURN
650 OUT 251,148: RETURN
651 OUT 251,201: RETURN      '35/70
652 OUT 251,211: RETURN
653 OUT 251,213: RETURN
654 OUT 251,178: RETURN
655 OUT 251,166: RETURN
656 OUT 251,197: RETURN
657 OUT 251,171: RETURN
658 OUT 251,234: RETURN
659 OUT 251,154: RETURN
660 OUT 251,203: RETURN
661 OUT 251,213: RETURN      '42/70
662 OUT 251,234: RETURN
663 OUT 251,211: RETURN
664 OUT 251,171: RETURN
665 OUT 251,203: RETURN
666 OUT 251,230: RETURN
667 OUT 251,218: RETURN
668 OUT 251,179: RETURN
669 OUT 251,221: RETURN
670 OUT 251,237: RETURN
671 OUT 251,235: RETURN      '49/70
```

```

672 OUT 251,183: RETURN
673 OUT 251,215: RETURN
674 OUT 251,219: RETURN
675 OUT 251,221: RETURN
676 OUT 251,222: RETURN
677 OUT 251,246: RETURN
678 OUT 251,238: RETURN
679 OUT 251,190: RETURN
680 OUT 251,213: RETURN
681 OUT 251,221: RETURN      '56/70
682 OUT 251,237: RETURN
683 OUT 251,215: RETURN
684 OUT 251,238: RETURN
685 OUT 251,254: RETURN
686 OUT 251,253: RETURN
687 OUT 251,251: RETURN
688 OUT 251,247: RETURN
689 OUT 251,239: RETURN
690 OUT 251,223: RETURN
691 OUT 251,255: RETURN      '63/70
692 OUT 251,255: RETURN
693 OUT 251,255: RETURN
694 OUT 251,191: RETURN
695 OUT 251,223: RETURN
696 OUT 251,239: RETURN
697 OUT 251,247: RETURN
698 OUT 251,251: RETURN
699 OUT 251,253: RETURN
700 OUT 251,254: RETURN
801 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 601,602,603,604,605,606,607,608,609,610
: NEXT Z: RETURN
802 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 611,612,613,614,615,616,617,618,619,620
: NEXT Z: RETURN
803 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 621,622,623,624,625,626,627,628,629,630
: NEXT Z: RETURN
804 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 631,632,633,634,635,636,637,638,639,640
: NEXT Z: RETURN
805 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 641,642,643,644,645,646,647,648,649,650
: NEXT Z: RETURN
806 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 651,652,653,654,655,656,657,658,659,660
: NEXT Z: RETURN
807 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 661,662,663,664,665,666,667,668,669,670
: NEXT Z: RETURN
808 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 671,672,673,674,675,676,677,678,679,680
: NEXT Z: RETURN
809 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 681,682,683,684,685,686,687,688,689,690
: NEXT Z: RETURN
810 FOR Z=1 TO 10: X=RND(10): ON X GOSUB 691,692,693,694,695,696,697,698,699,700
: NEXT Z: RETURN
1000 '*****MULTI-BRANCH FOR SHADINGS *****
1010 ON N-500 GOSUB 801,802,803,804,805,806,807,808,809,810
1020 RETURN

```

Dennis was surrounded by shading in figure 5-24 not only by the dot groups selected by DATA values ranging from 501 to 510 but also by the values -10, 127 for pure black and -10, 0 for pure white. New DATA lines before and after the original lines were added. A typical DATA line in the additions for the shading would have a sequence such as -10, 127, 510, 509, 508, 507, 506, 505, 504, 503, 502, 501, -10, 0, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, -10, 127. Sequences similar to the first half of this example were added to the original lines for shading on the left, and sequences like the latter half were added for the right-hand gradations.

There seems to be no problem intermixing these subroutine calls with the dot codes, negative repeat codes, and line-feed code in the flow of the program. This is provided, of course, that codes in the range of 501 to 510 are recognized by the program as being different from codes in the range of 0 to 127, negative values, and code 999. Line 125 takes care of that matter by stating `IF N>500 AND N<511 THEN GOSUB 1000`.

You might think of these subroutine calls as paintbrush strokes that could take many more forms than the shading levels in the spotlighting of Dennis. These could provide preprogrammed instead of random dot sequences, be of many different lengths, provide a wide variety of textures (not just graduated tints), and be inserted anywhere in a line of print.

We'll follow up this idea shortly, but first let's consider doing a whole drawing with just the shadings. This is what Mike Keller proposes, under the apt title of "The Printer as a Paintbrush" in *80 Micro* (June 1983). Using Epson MX-80 and NEC 8023A printers, he converts an eight-by-ten photograph into a bit-image-drawn picture. He starts by placing a grid over a tracing of the photograph, dividing its whole area into $\frac{1}{12}$ -inch squares. In several well-illustrated steps, he shows how each of those little squares can be given a number from one to six, representing six levels of shading ranging from near-white to near-black. This is quite a project, because the number of squares needing numbers amounts to 120 rows with 96 squares in each row, or 11,520 in all.

An interesting printout justifies the labor. Viewed from across the room, it looks very much like a photograph, and

up close it has an arty, cubist appearance that is fascinating. Another nice feature is that the printouts come out somewhere near the original size of the photograph with this technique, whereas even fairly ambitious drawings by a LINEDRAW type of program are much smaller.

Once you've done the work of figuring out the codes and entering them into DATA statements, you can get more mileage out of your labors with several manipulations of the image Keller suggests. Besides negative and mirror images, you can vary the contrast by eliminating intermediate shading values—for example, have every square come out black or white—as is done by space scientists with computer-enhanced digital photos. You can also soften some of the edges in a drawing by using partial squares, that is, assigning more than a single value to one of those little squares, so that the left half of a square is lighter in shading than the right half, for example.

Program 5-14 (CUBETINT) has several of these suggestions built into it. And by using ten shading levels similar to those surrounding Dennis, in addition to pure black and pure white, it provides twelve levels instead of the six gradations used by Keller. The program also has provisions for both fixed-length (subroutines 801–810) and variable-length (subroutine 1000) squares of shading. Since Keller has already presented programs for IBM-Epson and Apple-type printers, CUBETINT is designed for TRS-80 printers.

PROGRAM 5-14. CUBETINT. Drawing with small “cubes” of shading.

```

10 'PROGRAM 5-14: "CUBETINT" -- DRAWING WITH SHADING LEVELS
15 'Copyright 1985 by John Warner Davenport
20 'FOR TRS-80 PRINTERS AND TRS-80 MODEL III (USE BOOTHE'S PRINTER DRIVER)
30 DEFINT A-Z
40 LPRINT CHR$(30);CHR$(27);CHR$(20);          'CONDENSED PRINTING
50 FL=10                                          'FIXED LENGTH OF CUBE
100 FOR R=181 TO 266                            'LOOP FOR 85 ROWS
110   LPRINT CHR$(18);                          'ENTER GRAPHIC MODE
120   READ N
130   IF N=>0 AND N<12 THEN RANDOM: ON N+1 GOSUB 800,801,802,803,804,805,806,807
      ,808,809,810,811: GOTO 120                  'FIXED-LENGTH SHADING LEVELS
140   IF N=999 THEN GOTO 160
150   IF N>300 AND N<400 THEN GOSUB 1000 : GOTO 120 'VARIABLE-LENGTH SHADING
160   LPRINT CHR$(13);                          'CARRIAGE RETURN, LINE FEED
170 NEXT R

```

[illegible]

```

615 RANDOM: Y=RND(7)-1: E=2[Y: RETURN
616 E=0: RETURN
617 RANDOM: Y=RND(7)-1: E=2[Y: RETURN
618 E=0: RETURN
619 RANDOM: Y=RND(7)-1: E=2[Y: RETURN
620 E=0: RETURN      '-----
621 E=64: RETURN      '12/70
622 E=32: RETURN
623 E=16: RETURN
624 E=8: RETURN
625 E=4: RETURN
626 E=2: RETURN
627 E=1: RETURN
628 RANDOM: Y=RND(7)-1: E=2[Y: RETURN
629 E=20: RETURN
630 E=65: RETURN      '-----
631 E=73: RETURN      '21/70
632 E=34: RETURN
633 E=50: RETURN
634 E=20: RETURN
635 E=65: RETURN
636 E=36: RETURN
637 E=72: RETURN
638 E=18: RETURN
639 E=66: RETURN
640 E=40: RETURN      '-----
641 E=42: RETURN      '28/70
642 E=21: RETURN
643 E=22: RETURN
644 E=52: RETURN
645 E=73: RETURN
646 E=84: RETURN
647 E=81: RETURN
648 E=38: RETURN
649 E=34: RETURN
650 E=20: RETURN      '-----
651 E=73: RETURN      '35/70
652 E=101: RETURN
653 E=84: RETURN
654 E=38: RETURN
655 E=50: RETURN
656 E=81: RETURN
657 E=106: RETURN
658 E=43: RETURN
659 E=105: RETURN
660 E=85: RETURN      '-----
661 E=83: RETURN      '42/70
662 E=43: RETURN
663 E=101: RETURN
664 E=106: RETURN
665 E=105: RETURN
666 E=51: RETURN

```



```

667 E=45: RETURN
668 E=38: RETURN
669 E=93: RETURN
670 E=91: RETURN      '-----
671 E=107: RETURN     '49/70
672 E=118: RETURN
673 E=117: RETURN
674 E=109: RETURN
675 E=93: RETURN
676 E=61: RETURN
677 E=55: RETURN
678 E=59: RETURN
679 E=62: RETURN
680 E=85: RETURN      '-----
681 E=93: RETURN     '56/70
682 E=91: RETURN
683 E=117: RETURN
684 E=59: RETURN
685 E=63: RETURN
686 E=95: RETURN
687 E=111: RETURN
688 E=119: RETURN
689 E=123: RETURN
690 E=125: RETURN     '-----
691 E=127: RETURN     '63/70
692 E=127: RETURN
693 E=127: RETURN
694 E=126: RETURN
695 E=125: RETURN
696 E=123: RETURN
697 E=119: RETURN
698 E=111: RETURN
699 E=95: RETURN
700 E=63: RETURN      '-----
800 LPRINT STRING$(FL,128);: RETURN      'BLANK-WHITE SPACE
801 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 601 ,602 ,603 ,604 ,605 ,606 ,607
      ,608 ,609 ,610 : LPRINT CHR$(E+128);: NEXT Z: RETURN
802 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 611 ,612 ,613 ,614 ,615 ,616 ,617
      ,618 ,619 ,620 : LPRINT CHR$(E+128);: NEXT Z: RETURN
803 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 621 ,622 ,623 ,624 ,625 ,626 ,627
      ,628 ,629 ,630 : LPRINT CHR$(E+128);: NEXT Z: RETURN
804 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 631 ,632 ,633 ,634 ,635 ,636 ,637
      ,638 ,639 ,640 : LPRINT CHR$(E+128);: NEXT Z: RETURN
805 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 641 ,642 ,643 ,644 ,645 ,646 ,647
      ,648 ,649 ,650 : LPRINT CHR$(E+128);: NEXT Z: RETURN
806 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 651 ,652 ,653 ,654 ,655 ,656 ,657
      ,658 ,659 ,660 : LPRINT CHR$(E+128);: NEXT Z: RETURN
807 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 661 ,662 ,663 ,664 ,665 ,666 ,667
      ,668 ,669 ,670 : LPRINT CHR$(E+128);: NEXT Z: RETURN
808 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 671 ,672 ,673 ,674 ,675 ,676 ,677
      ,678 ,679 ,680 : LPRINT CHR$(E+128);: NEXT Z: RETURN
809 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 681 ,682 ,683 ,684 ,685 ,686 ,687
      ,688 ,689 ,690 : LPRINT CHR$(E+128);: NEXT Z: RETURN

```

```

810 FOR Z=1 TO FL: X=RND(10): ON X GOSUB 691 ,692 ,693 ,694 ,695 ,696 ,697
,698 ,699 ,700 : LPRINT CHR$(E+128);: NEXT Z: RETURN
811 LPRINT STRING$(FL,255);: RETURN 'SOLID-BLACK PRINTING
1000 '***** MULTI-BRANCH FOR VARIABLE-LENGTH SHADINGS *****
1010 'UP TO 9 DOT-COLUMNS (DETERMINED BY MIDDLE DIGIT OF A 300-SERIES DATA CODE)
OF ANY OF SHADINGS 1-9 (DETERMINED BY LAST DIGIT) MAY BE SELECTED BY CODE
S RANGING FROM 301 TO 399
1020 RANDOM
1030 N$=STR$(N): FL=VAL(MID$(N$,3,1))
1040 ON VAL(RIGHT$(N$,1)) GOSUB 801,802,803,804,805,806,807,808,809
1050 FL=10
1060 RETURN

```

In the digitizing, a black-and-white photocopy of an eighty-by-ten cosmetics advertisement in color was divided into eighty-five rows of sixty-four squares each, the square size being $\frac{1}{8}$ inch. (Using $\frac{1}{8}$ inch instead of Keller's $\frac{1}{12}$ inch largely canceled out the advantages of having twelve instead of six shading levels, as it turned out.) On the DMP-400, each of these squares was printed out as a seven-by-ten dot matrix in condensed pitch (approximately $\frac{1}{10}$ inch square), making the printout about 20% smaller than the original. The twelve degrees of shading were represented by codes ranging from 0 (pure white) to 11 (pure black).

Figure 5-25 shows part of the coding sheet. Notice that while most squares were given a single code in the 0-to-11 range, some were divided (diagonally) to provide for two different levels of shading over the ten dot columns of a

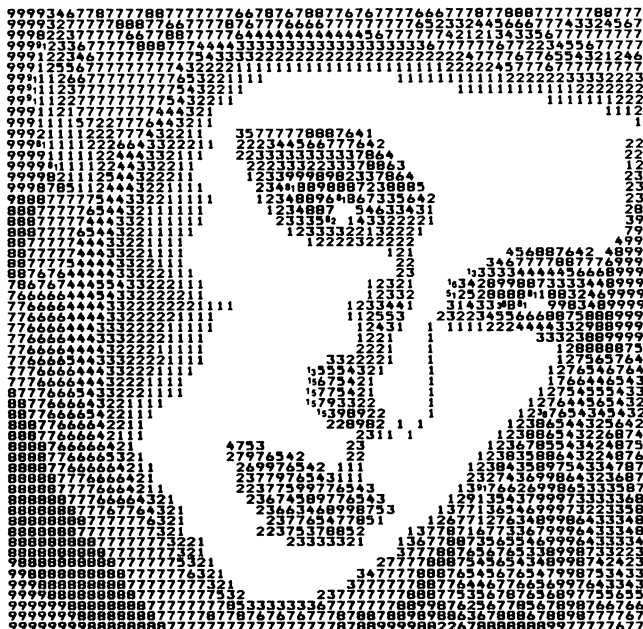


FIGURE 5-25. Digitizing a photograph for a program (CUBETINT) that draws pictures with squares of shading instead of lines. Most digits represent the shading value of a ten-column square. Where two smaller digits appear, combinations of shading values that sum to ten dot columns are printed. (Original photograph courtesy of Max Factor.)

square. In the program, code numbers in the range of 301 to 399 are sent (by line 150) to subroutine 1000, which uses string functions to determine the middle and last digits of the code. The middle digit, which can range from 1 to 9, determines how many dot columns will be used, and the last digit determines what level of shading will be in those dot columns. (Shading levels are restricted to levels 1 to 9 in this subroutine, that is, they do not include pure white or heavy black.) Every time one 300-series code is used, it must be paired with another one, and the middle digits of the two codes must add up to ten, the standard number of dot columns in a square.

A “normal” printout from the CUBETINT program is shown in figure 5-26. This is one in which the full range of shadings is used in an attempt to get a normal degree of contrast in the drawing.

By changing the subroutine numbers in lines 130 and 1040, you can vary the contrast many different ways. Eliminating extreme shading values can be done by not calling subroutines 800, 801, 802, 809, 810, and 811 at all, for example, so that the lightest shading level is 3 and the darkest is

FIGURE 5-26. Rendering of an eight-by-ten photograph by the CUBETINT program (reduced 50% from actual printout size on the DMP-400).





FIGURE 5-27. Printouts from the CUBETINT program: horizontal mirror images (upper pair), negative and positive images (middle pair), and high and low contrast (bottom pair).

8. This produces the lowest-contrast printout shown in figure 5-27 (bottom right panel). A high-contrast printout, produced by eliminating all intermediate shading levels, is also shown (bottom left). There is also a negative image (center left), generated simply by reversing the orders of the original subroutine numbers in lines 130 and 1040.

For a horizontal mirror image, the DATA values for each row can be read into an array and then printed from the array in reverse order. This would mean adding `DIM N(64)` early in the program, changing each `N` to `N(I)`, and putting a new

FOR-NEXT loop inside the first loop. At the beginning of the new loop, `FOR I=1 TO 64` would produce a normal print-out and `FOR I=64 TO 1 STEP -1` would result in a mirror-image rendition. Using `DIM N(64)`, incidentally, assumes that all partial squares are combined into regular ten-dot column squares, so that each row will contain only sixty-four DATA values; if this is not done, a higher value than 64 will be needed in the DIM statement, and the programming will be more complicated in other ways.

Lines and textures combined. We can come even closer to “photographic” pictures if we combine regular line drawing with these paintbrush strokes we call shading levels. Instead of using the shading just to surround a line drawing, as in the spotlighting of Dennis, or having the whole drawing done by shading blocks, as with the CUBETINT program, we can integrate the lines and the shading. A few additions to the LINEDRAW programs are all we need, as program 5-15 shows.

This program, called FOTOGRAF, has the usual LINEDRAW features of immediately printing any DATA code in the range of 0 to 127 (line 150), repeating the printing of any code that is preceded by a negative number (lines 160–180), executing a line feed and carriage return at the end of each print line (lines 100 and 190), and counting bytes in a print line (lines 150, 180, 190, 1040, 2070, and 3035). It also has provisions for printing a ten-column (fixed-length) block of shading (line 110 and subroutines 1000, 800, and 600). And as with CUBETINT, the block of shading can be split into partial blocks having different shading values (line 120 and subroutine 2000).

Still another feature, not used in the previous drawing programs, is a backspacing technique (line 130 and subroutine 3000). The idea here is to make it possible to lay down a strip of shading (usually light) and then back up in order to draw something on top of it. Or the reverse: adding shading after having drawn some sharp details in the usual LINEDRAW fashion. One way or another, FOTOGRAF offers you a total intermixture of high-resolution lines and curves with textures.

Our example for this program’s output is the same as for CUBETINT—except that only the facial portion of the

young woman's photo was digitized for a FOTOGRAF version. Figure 5-28 shows the printout. Now, it's safe to say, we're getting a lot closer to photography than we did with CUBETINT, even when printouts from the latter program are reduced (compare figure 5-28 with figures 5-26 and 5-27).

PROGRAM 5-15. FOTOGRAF. Combining line drawing with shading levels for "photographic" drawings.

```

10 'PROGRAM 5-15: "FOTOGRAF" -- LINE DRAWING WITH SHADING LEVELS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 MODEL 4 AND TRS-80 PRINTERS
30 DEFINT A-Z
40 FL=10
50 FOR R=10000 TO 10430 STEP 10
60 C=0
70 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18);
90 READ N: IF N<0 THEN GOTO 160
100 IF N=999 THEN GOTO 185
110 IF N>500 AND N<999 THEN GOSUB 1000 : GOTO 90      'FIXED-LENGTH (10 DOT-COLUM
    NS) SHADING
120 IF N>300 AND N<400 THEN GOSUB 2000 : GOTO 90      'VARIABLE-LENGTH SHADING
130 IF N>400 AND N<481 THEN GOSUB 3000 : GOTO 90      'BACKSPACE SUBROUTINE
140 IF N=9 THEN N=8                                  'DETOUR (TROUBLE CODE)
150 IF N>=0 AND N<128 THEN LPRINT CHR$(N+128);: C=C+1: PRINT C;: GOTO 90
160 READ M      'CODE # WAS NOT POSITIVE OR ZERO, SO READ NEXT
165 IF M=127 AND C>389 THEN M=0
170 IF M=9 THEN M=8
180 LPRINT STRING$(-N,M+128);: C=C+ABS(N): PRINT C;: GOTO 90
185 OE$=MID$(STR$(R),5,1): OE=VAL(OE$)
188 IF OE=1 OR OE=3 OR OE=5 OR OE=7 OR OE=9 THEN T=3 ELSE T=8
190 LPRINT CHR$(30);STRING$(T,32);" ";R;"          ";C;: FOR Q=1 TO 20: LPRINT CHR$(
    27);CHR$(51);: NEXT Q: LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(0);: PRINT: PRI
    NT "BYTE COUNT FOR ROW";R;" = ";C      'END-OF-ROW PRINTS, LINE FEED, CARR. RET
    URN
200 NEXT R
600 '=====CODES FOR RANDOM SELECTIONS=====
601 E=0: RETURN      '1/70
602 E=0: RETURN
603 E=0: RETURN
604 E=0: RETURN
605 E=0: RETURN
606 E=0: RETURN
607 E=0: RETURN
608 RANDOM: Y=RND(7)-1: E=2^Y: RETURN
609 E=0: RETURN
610 E=0: RETURN      '-----
611 E=0: RETURN      '5/70
612 RANDOM: Y=RND(7)-1: E=2^Y: RETURN
613 RANDOM: Y=RND(7)-1: E=2^Y: RETURN
614 E=0: RETURN

```

```

.
.   (AS IN 'CUBETINT')
.
695 E=125: RETURN
696 E=123: RETURN
697 E=119: RETURN
698 E=111: RETURN
699 E=95: RETURN
700 E=63: RETURN      '-----
800 'ROUTING FOR FIXED-LENGTH SHADINGS
810 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 601 ,602 ,603 ,604 ,605 ,6
    06 ,607 ,608 ,609 ,610 : LPRINT CHR$(E+128);: NEXT Z: RETURN
820 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 611 ,612 ,613 ,614 ,615 ,6
    16 ,617 ,618 ,619 ,620 : LPRINT CHR$(E+128);: NEXT Z: RETURN
.
.   (AS IN 'CUBETINT')
.
890 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 681 ,682 ,683 ,684 ,685 ,6
    86 ,687 ,688 ,689 ,690 : LPRINT CHR$(E+128);: NEXT Z: RETURN
900 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 691 ,692 ,693 ,694 ,695 ,6
    96 ,697 ,698 ,699 ,700 : LPRINT CHR$(E+128);: NEXT Z: RETURN
1000 'MULTI-BRANCH FOR FIXED-LENGTH SHADINGS
1010 RANDOM
1020 LPRINT CHR$(18);
1030 ON N-500 GOSUB 810 ,820 ,830 ,840 ,850 ,860 ,870 ,880 ,890 ,900
1040 C=C+10: PRINT C;
1050 RETURN
2000 'MULTI-BRANCH FOR VARIABLE-LENGTH SHADINGS
2010 'UP TO 9 DOT-COLUMNS (DETERMINED BY MIDDLE DIGIT OF A 300-SERIES DATA CODE
    ) OF ANY OF SHADINGS 1-10 (DETERMINED BY LAST DIGIT) MAY BE SELECTED BY COD
    ES RANGING FROM 301 TO 399
2020 RANDOM
2030 LPRINT CHR$(18);
2040 N$=STR$(N): FL=VAL(MID$(N$,3,1))
2050 IF RIGHT$(N$,1)="0" THEN GOSUB 900 : GOTO 2080
2060 ON VAL(RIGHT$(N$,1)) GOSUB 810 ,820 ,830 ,840 ,850 ,860 ,880 ,880 ,
    890
2070 C=C+FL: PRINT C;
2080 FL=10
2090 RETURN
3000 'SUBROUTINE FOR BACKSPACING WITHIN A PRINT LINE
3010 PRINT "BACKSPACE CODE:";N;" BYTE TOTAL:";C
3020 N$=STR$(N): BK=20*VAL(RIGHT$(N$,2))
3030 LPRINT CHR$(30);CHR$(8);CHR$(BK);CHR$(18);
3035 C=C-BK/2: PRINT C
3040 RETURN
10000 REM -- DATA
10001 DATA 501,501,501,351,322,333,502,502,502,403,48,112,80,112,-2,80,-2,112,48
    ,80,-3,112,-4,80,48,96,64,-2,64,0,-2,64,-5,0,501,501,501,501,501
10002 DATA 501,501,501,501,501,501,501,501
10004 DATA 501,501,501,501,501,501,501,501
10006 DATA 501,501,501,501,501,501,501,501
10008 DATA 501,501,501
10009 DATA -3,127,999

```

```

10010 DATA 501,501,501,322,383,503,401,-3,0,1,3,5,14,11,31,27,507,507,357,126,11
      8,116,124,114,502,501,402,116,104,80,56,80,40,48,96,80,80,80,64,-3,32,-4,6
      4,0,501,501
10012 DATA 501,501,501,501,501,501,501,501
10014 DATA 501,501,501,501,501,501,501,501
10016 DATA 501,501,501,501,501,501,501,501
10018 DATA 501,501,501
10019 DATA -3,127,999
10020 DATA 501,501,371,332,503,503,503,21,2,3,7,11,78,29,46,43,119,507,507,507,3
      57,48,90,52,88,81,40,96,18,80,80,96,32,4,64,64
.
.   (ETC.)
.

```

We're still not all the way there—you can easily tell that figure 5-28 is not a photograph, and there's little danger that printouts from a FOTOGRAF program for seven- or eight-pin printers will be mistaken for true photographs. The dots are too big for subtle textures, there aren't enough shading levels for realistic skin tones, and the RND function isn't random enough for top-quality shadings.

Nevertheless, despite the limitations, the DMP-400 can manage eyes, reflections of light off eyes and lips, eyelashes, and probably contrasting strands of hair fairly well. With an Epson FX-80 or RX-80, we could use the 240-dots-per-inch density and undoubtedly come up with something better. But that might not be worth the extra work in digitizing (2.4 times as many codes required), because the increase in resolution would still be only in the horizontal direction. If you look closely at some of the eyelashes in figure 5-28, you'll see that we need increased vertical resolution just as much. Perhaps a printer such as the Epson LQ-1500, which has 24 pins packed into the same printhead space as the FX-80's top eight pins as well as the 240-dots-per-inch density, could produce a printout that would pass as a true photograph.

With seven- or eight-pin printers, you can come closer than figure 5-28's example if you choose an easier subject to simulate. Dark, high-contrast photos are much easier than well-lighted subjects with small changes in texture, especially if the latter are in color instead of black and white. For a dramatic comparison, try the famous *Life* magazine photograph taken by Karsh showing a scowling Winston Churchill in high contrast, and put your FOTOGRAF printout next to the several renditions of this photograph that have appeared in



FIGURE 5-28. Simulation of a photograph using the FOTOGRAF program. The subject is the same as the young woman in figures 5-26 and 5-27.

microcomputer magazines and printer manuals in the past three years.

The drawing in figure 5-28 was printed in forty-two passes of the printhead, and each print line had 390 dot patterns. (That's 16,380 dot columns.) The coding was for the DMP-400's highest density, 100 dots per inch. As before, the coding was done with a hand-and-eye enlargement of the original photograph, and in this case the coding sheet was thirty by thirty-nine inches in size.

The early DATA lines of the FOTOGRAF program show examples of the various drawing features it provides. Nearly all of the lines contain 500-series codes for ten-column squares of shading and 300-series codes for under-ten-column blocks of shading, as in the CUBETINT program. These, which reduce the digitizing labor considerably, are intermixed with LINEDRAW-type codes for individual dot columns with no difficulty. You'll also see a few examples of 400-series codes for backspacing and overprinting; notice that DATA statements containing these have to have ten more codes for each ten-column unit of backspacing for the overprinting. So the total number of dot codes required is more than the 16,380 dot columns they fill.

Keep in mind that drawings by the FOTOGRAF program don't have to be done in a single print run. Following Kalinowski's suggestion (*80 Micro*, November 1983), you can also program different parts of a figure for separate printings that start at the same point on the paper. In his case, this meant separate printings for different ribbon colors, which opens up a whole new world of printer graphics. But even in black-and-white graphics you can find advantages of, for example, separating the LINEDRAW features of a drawing from the shadings provided by 300- and 500-series codes. Doing this does require careful planning, though, and it also involves a certain amount of gambling on the consistency of line feeds, since you are trying to print one printout on top of another with an exact fit.

As with the CUBETINT digitizing, it's nice to discover that there are several ways of making use of the coding besides just printing a four-by-four-inch picture like the printout in figure 5-28. For example, with a widebody printer you can use the FOTOGRAF codes to print a picture measuring about twelve inches square, or you might have a two-by-

three-inch printout that you'd print in poster size of, say, thirteen by twenty inches. This is one of the things that program 5-16 (ENLARGER) does.

PROGRAM 5-16. ENLARGER. Making poster-size printouts with digitized drawings.

```

10 'PROGRAM 5-16: "ENLARGER" -- MAKING POSTER-SIZE PICTURES
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 DMP-400 PRINTER AND MODELS III OR 4
30 CLEAR 5000                                '(NOT NECESSARY FOR MODEL 4)
35 DEFINT A-Z
40 INPUT "NUMBER OF DOT-COLUMNS PER ROW";CL
50 INPUT "NUMBER OF ROWS";RW
60 INPUT "FIXED LENGTH OF SHADING";FL
70 DIM N(2*CL),X(7),Y(7),BYTE$(2*CL)
80 PR$=STRING$(2,156) 'PRINTED RECTANGLE (ADJUST SIZE FOR DEGREE OF ENLARG'T)
90 BK$=STRING$(2,128) 'BLANK SPACE OF SAME HEIGHT & WIDTH AS RECTANGLE
100 LF$=CHR$(27)+CHR$(50)+CHR$(27)+CHR$(50)+CHR$(27)+CHR$(50) '3/72" LINE FEED
110 'NOTE: LINE FEED SIZE MUST EQUAL HEIGHT OF RECTANGLE FOR BUTTING OF LINES
120 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
130 LPRINT CHR$(30);CHR$(27);CHR$(23);CHR$(27);CHR$(31); 'ELITE, BOLD
140 LPRINT CHR$(18); 'ENTER GRAPHIC MODE
150 REM -- LINES 160-230 STORE DECIMAL DOT-CODES IN N(C) ARRAYS
155 PRINT "STORING CODES IN ARRAY"
160 FOR R=1 TO RW
170   C=1
180   READ N
185   IF N=999 THEN N(C)=N: PRINT "999": C=C+1: NUM=C: GOTO 240
190   IF N>=0 AND N<128 THEN N(C)=N: C=C+1: GOTO 230
200   IF N<0 THEN READ M: FOR X=1 TO -N: N(C)=M: PRINT M; C=C+1: NEXT X: GOTO 2
30
210   IF N>500 AND N<600 THEN GOSUB 1000: GOTO 230 'FIXED-LENGTH SHADING
220   IF N>300 AND N<400 THEN GOSUB 2000: GOTO 230 'VARIABLE-LENGTH SHADING
225   IF N>400 AND N<411 THEN N(C)=N: C=C+1: GOTO 230
230   GOTO 180
240 REM -- LINES 250-340 CONVERT DECIMAL CODES TO BINARY STRINGS
245 PRINT "CONVERTING TO BINARY CODES"
250 FOR C=1 TO NUM
260   IF N(C)=999 THEN GOTO 350
265   IF N(C)>400 AND N(C)<411 THEN N(C)=N(C)+1: B$=RIGHT$(STR$(N(C)),1): BYTE
$(C)=B$+B$+B$+B$+B$+B$: C=C+1: GOTO 340
270   NDEC=N(C)
280   FOR P=6 TO 0 STEP -1
290     X(P)=NDEC-2^P '(MODEL III: LEFT BRACKET FOR "^")
300     IF X(P)<0 THEN Y$(P)="0": GOTO 320
310     Y$(P)="1": NDEC=X(P)
320   NEXT P
330   GOSUB 470 'ASSEMBLE AND DISPLAY BINARY-FORM BYTE$(C)
340 NEXT C
350 REM -- LINES 360-430 PRINT BITS STORED IN BYTE$(C) ARRAY
355 PRINT "PRINTING BITS OF BYTE$ ARRAY"

```

```

360 FOR L=1 TO 7
370 FOR C=1 TO NUM
375 DIG$=MID$(BYTE$(C),L,1): BK=40*VAL(DIG$)-40
377 IF VAL(DIG$)>1 THEN LPRINT CHR$(30);CHR$(8);CHR$(BK);CHR$(18);: GOTO 4
    20
380 IF BW$="B" AND DIG$="0" THEN LPRINT BK$;: GOTO 420
390 IF BW$="B" AND DIG$="1" THEN LPRINT PR$;: GOTO 420
400 IF BW$="W" AND DIG$="0" THEN LPRINT BK$;
410 IF BW$="W" AND DIG$="1" THEN LPRINT PR$;
420 NEXT C
430 LPRINT LF$;CHR$(27);CHR$(16);CHR$(0);CHR$(0); 'LF AND CR
440 NEXT L
450 NEXT R
460 END
470 '***** ASSEMBLY OF BYTE$(C) *****
480 BYTE$(C)=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6)
490 RETURN
600 '=====CODES FOR RANDOM SELECTIONS=====
601 E=0: RETURN '1/70
602 E=0: RETURN
603 E=0: RETURN
604 E=0: RETURN
605 E=0: RETURN
606 E=0: RETURN
607 E=0: RETURN
608 RANDOM: Y=RND(7)-1: E=2^Y: RETURN
609 E=0: RETURN
610 E=0: RETURN '-----
611 E=0: RETURN '5/70
612 RANDOM: Y=RND(7)-1: E=2^Y: RETURN
.
. (AS IN 'CUBETINT' OR 'FOTOGRAF')
.
695 E=125: RETURN
696 E=123: RETURN
697 E=119: RETURN
698 E=111: RETURN
699 E=95: RETURN
700 E=63: RETURN '-----
800 'ROUTING FOR FIXED-LENGTH SHADINGS
810 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 601 ,602 ,603 ,604 ,605 ,6
    06 ,607 ,608 ,609 ,610 : N(C)=E: C=C+1: NEXT Z: RETURN
820 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 611 ,612 ,613 ,614 ,615 ,6
    16 ,617 ,618 ,619 ,620 : N(C)=E: C=C+1: NEXT Z: RETURN
830 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 621 ,622 ,623 ,624 ,625 ,6
    26 ,627 ,628 ,629 ,630 : N(C)=E: C=C+1: NEXT Z: RETURN
840 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 631 ,632 ,633 ,634 ,635 ,6
    36 ,637 ,638 ,639 ,640 : N(C)=E: C=C+1: NEXT Z: RETURN
850 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 641 ,642 ,643 ,644 ,645 ,6
    46 ,647 ,648 ,649 ,650 : N(C)=E: C=C+1: NEXT Z: RETURN
860 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 651 ,652 ,653 ,654 ,655 ,6
    56 ,657 ,658 ,659 ,660 : N(C)=E: C=C+1: NEXT Z: RETURN
870 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 661 ,662 ,663 ,664 ,665 ,6
    66 ,667 ,668 ,669 ,670 : N(C)=E: C=C+1: NEXT Z: RETURN

```

```

880 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 671 ,672 ,673 ,674 ,675 ,6
76 ,677 ,678 ,679 ,680 : N(C)=E: C=C+1: NEXT Z: RETURN
890 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 681 ,682 ,683 ,684 ,685 ,6
86 ,687 ,688 ,689 ,690 : N(C)=E: C=C+1: NEXT Z: RETURN
900 FOR Z=1 TO FL: RANDOM: X=RND(10): ON X GOSUB 691 ,692 ,693 ,694 ,695 ,6
96 ,697 ,698 ,699 ,700 : N(C)=E: C=C+1: NEXT Z: RETURN
1000 'MULTI-BRANCH FOR FIXED-LENGTH SHADINGS
1010 RANDOM
1020 LPRINT CHR$(18);
1030 ON N-500 GOSUB 810 ,820 ,830 ,840 ,850 ,860 ,870 ,880 ,890 ,900
1050 RETURN .
2000 'MULTI-BRANCH FOR VARIABLE-LENGTH SHADINGS
2010 'UP TO 9 DOT-COLUMNS (DETERMINED BY MIDDLE DIGIT OF A 300-SERIES DATA CODE
) OF ANY OF SHADINGS 1-10 (DETERMINED BY LAST DIGIT) MAY BE SELECTED BY COD
ES RANGING FROM 301 TO 399
2020 RANDOM
2030 LPRINT CHR$(18);
2040 N$=STR$(N): FL=VAL(MID$(N$,3,1))
2050 IF RIGHT$(N$,1)="0" THEN GOSUB 900 : GOTO 2080
2060 ON VAL(RIGHT$(N$,1)) GOSUB 810 ,820 ,830 ,840 ,850 ,860 ,880 ,880 ,
890
2080 FL=10
2090 RETURN
10000 REM -- DATA
10001 DATA 501,501,501,351,322,333,502,502,502,403,48,112,80,112,-2,80,-2,112,48
,80,-3,112,-4,80,48,96,64,-2,64,0,-2,64,-5,0,501,501,501,501,501
10002 DATA 501,501,501,501,501,501,501,501
10004 DATA 501,501,501,501,501,501,501,501
10006 DATA 501,501,501,501,501,501,501,501
10008 DATA 501,501,501
10009 DATA -3,127,999
10010 DATA 501,501,501,322,383,503,401,-3,0,1,3,5,14,11,31,27,507,507,357,126,11
8,116,124,114,502,501,402,116,104,80,56,80,40,48,96,80,80,80,64,-3,32,-4,64
,0,501,501
10012 DATA 501,501,501,501,501,501,501,501
10014 DATA 501,501,501,501,501,501,501,501
10016 DATA 501,501,501,501,501,501,501,501
10018 DATA 501,501,501
10019 DATA -3,127,999
10020 DATA 501,501,371,332,503,503,503,21,2,3,7,11,78,29,46,43,119,507,507,507,3
57,48,90,52,88,81,40,96,18,80,80,96,32,4,64,64
.
. (ETC.)
.

```

The main purpose of ENLARGER is to provide blowups of drawings, including LINEDRAW- as well as FOTO-GRAF-produced drawings. The principle involved is about the same as the use of different-sized squares to represent pixels in screen dumps (see program 4-7, DUMPSIZE). But in this case we are representing each dot printed by the printer (instead of screen pixels) by larger graphic prints.

These larger prints need not be squares. We can use rectangles for the enlarging, and in so doing we can change the height-to-width ratio. This makes possible a second function of the program—to make the original coding appropriate for a different printer or a different print density (usually both). For instance, if you wanted to use the TRS-80 codes for a large printout of Dennis (program A-1 in appendix A) or Dagwood (program 5-10) on an IBM-Epson printer in double density, you could adjust for the TRS-80's 100-DPI density by using a nearly-square rectangle instead of a perfect square. In shifting from TRS-80 to Epson hardware, it is also necessary to invert all the original dot codes to their mirror images. This conversion is easily accomplished by manipulating the order of subscripts for the `BYTE$(C)` array in line 480 of the `ENLARGER` program.

When these poster-size printouts are viewed from a distance of several yards, they certainly look like photographs and like posters of the sort that make political convention halls seem rather cluttered. When you get a little closer, the dots used in light shading look more like freckles than dots because they're now rectangular, and they seem to be more unvarying in size than in the original small printouts.

If this bothers you, or if you want to try converting the portrait into something more abstract, you can represent the original dots by something other than solid rectangles—ellipses, diamonds, crosses, open rectangles or squares, and other possibilities—and not worry so much about the height-to-width ratio.

Those interested in mere enlargement will find that many different sizes of enlarged drawings are possible, depending on the size of the graphic element representing the original dots. To get an enlargement that makes the most of an eight-inch printer, you'll have to experiment with other sizes than our three-dot-high element in program 5-16. The shape as well will need adjustment, depending importantly on the printing density you select, and the line feed will need to be set equal to the height of the element. If there is no need for converting codes to mirror-image codes, you need only reverse the order of the subscripts of `BYTE$()` in line 480.

At the present stage of development of picture drawing with dot-matrix printers, the manual digitizing labor for just one drawing can be enormous—many dozens of hours, in

fact, if you are a novice. As a result, programs such as FOTOGRAF are likely to be enjoyed more by hobbyists than by professionals. For the dedicated hobbyist this makes it all the more fun—like doing one of those five-thousand-piece jigsaw puzzles. People in more of a hurry, however, may not have to wait long for new devices (e.g., optical scanners) that will reduce coding time sharply without a sacrifice of visual quality.

Your Own Graphics Catalogue

CHARTPAK . . . Letraset . . . Formatt . . . Zipatone . . . do these names ring a bell? You'd certainly know them if you were a graphic artist or draftsman, because these (and many others) are the companies that produce the plastic tapes, screens, and rub-on lettering sheets that artisans in the world of commercial art and illustration depend on daily.

After the last chapter, you might be wondering: If the dot-matrix printer can draw so well, could it take the place of those graphic arts materials? How nearly can the BASIC-controlled printer duplicate the wares in the catalogues put out by Chartpak, Letraset, et al.? Well, let's see.

How to Make Graphic Art and Drafting "Tapes"

Drafting tapes. From the examples of horizontal lines in chapter 5 we've already seen how the microcomputer and printer can make solid, dashed, dotted, and patterned lines of varying thickness. Figure 6-1 has some more examples, showing that even where the patterns get pretty fancy in the catalogues the printer keeps up quite well.

As with the catalogue tapes, these printer-produced lines can be used as connecting lines in graphs, as borders or frames in layouts, and other ways. The BASIC statements for them may be part of a larger program for a graph, diagram, or table in which one of these patterns is a horizontal line, or

DRAFTING TAPES

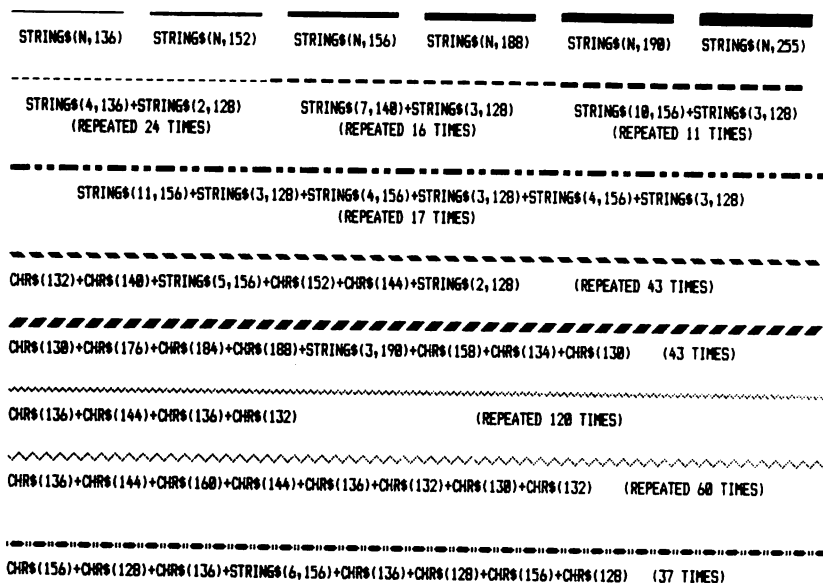


FIGURE 6-1. Simulated drafting tapes and their BASIC statements for TRS-80 printers. Where “REPEATED . . . TIMES” occurs, the LPRINT statements are enclosed in FOR-NEXT loops.

the printed lines may be cut out and pasted up as vertical, horizontal, or sloping lines.

Remember, too, that the lines may be printed on transparent sheets placed in the printer as well as on regular white paper. Thus a good imitation of transparent drafting tapes can be used for pasteups. And unlike the real tapes, you don’t have to worry about running out. (But you might have to worry about wearing out—the printer’s ribbon, that is.)

The examples in figure 6-1 were from the DMP-400. Again, for IBM-Epson systems, use a chart (figure 4-1 or 4-3) to convert from TRS-80 dot codes. For Apple printers, subtract 128 from the Radio Shack dot codes.

Statistical tapes. With two or three sweeps of the printhead, the printer will put out a nice variety of tapes for statistical charts. Figure 6-2 shows some samples that are not exact replicas of tapes in the catalogues but ones that will serve just as well. These usually look best when they are printed in bold, except for the fine shading in the second example—depending on how well inked the ribbon is, bold printing can often produce a muddy-looking appearance with such

STATISTICAL TAPES

PRINTED IN TWO PASSES:



```
230 A$=CHR$(231)+CHR$(179)+CHR$(153)+CHR$(205)
240 FOR N=1 TO 50: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
250 B$=CHR$(204)+CHR$(230)+CHR$(243)+CHR$(217)
260 FOR N=1 TO 50: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



```
270 A$=CHR$(213)+CHR$(163)+CHR$(213)+CHR$(137)
280 FOR N=1 TO 50: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
290 B$=CHR$(234)+CHR$(196)+CHR$(234)+CHR$(209)
300 FOR N=1 TO 50: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



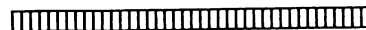
```
310 A$=CHR$(213)+CHR$(129)+CHR$(171)+CHR$(129)
320 FOR N=1 TO 50: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
330 B$=CHR$(234)+CHR$(192)+CHR$(213)+CHR$(192)
340 FOR N=1 TO 50: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



```
350 A$=CHR$(135)+CHR$(195)+CHR$(225)+CHR$(177)+CHR$(153)+CHR$(141)
360 FOR N=1 TO 33: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
370 B$=CHR$(195)+CHR$(225)+CHR$(240)+CHR$(216)+CHR$(204)+CHR$(198)
380 FOR N=1 TO 33: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



```
390 A$=STRING$(4,255)+CHR$(129)
400 FOR N=1 TO 40: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
410 B$=STRING$(4,255)+CHR$(192)
420 FOR N=1 TO 40: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



```
430 A$=CHR$(255)+STRING$(4,129)
440 FOR N=1 TO 40: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
450 B$=CHR$(255)+STRING$(4,192)
460 FOR N=1 TO 40: LPRINT B$;: NEXT N: LPRINT CHR$(255)
```



```
470 A$=STRING$(200,219)
480 LPRINT A$;CHR$(255);LF$;
490 B$=STRING$(200,237)
500 LPRINT B$;CHR$(255)
```

PRINTED IN THREE PASSES:



```
520 A$=CHR$(213)+CHR$(137)+CHR$(213)+CHR$(163)
530 FOR N=1 TO 50: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
540 B$=CHR$(170)+CHR$(145)+CHR$(170)+CHR$(196)
550 FOR N=1 TO 50: LPRINT B$;: NEXT N: LPRINT CHR$(255);LF$;
560 C$=CHR$(213)+CHR$(226)+CHR$(213)+CHR$(200)
570 FOR N=1 TO 50: LPRINT C$;: NEXT N: LPRINT CHR$(255)
```



```
580 A$=CHR$(145)+CHR$(129)+CHR$(197)+CHR$(129)
590 FOR N=1 TO 50: LPRINT A$;: NEXT N: LPRINT CHR$(255);LF$;
600 B$=CHR$(162)+CHR$(128)+CHR$(136)+CHR$(128)
610 FOR N=1 TO 50: LPRINT B$;: NEXT N: LPRINT CHR$(255);LF$;
```

FIGURE 6-2. Statistical tapes simulated by repeating groups of dot codes in two or three printhead passes. The programming is for TRS-80 printers.

fineness or may be difficult to photograph for camera-ready copy.

Borders. Simple borders like those in the catalogues can be programmed for the dot-matrix printer using either bit-image codes in graphic mode or the ready-made block-graphic characters (if available) in text mode. In either case, as figure 6-3 illustrates, FOR-NEXT statements surrounding code sequences make it easy. The sequences are usually shorter when you use the block-graphic characters, but you are less limited in what you can do when you use the bit-image dot patterns.

BORDERSTEXT MODEBLOCK-GRAPHIC CODE SEQUENCES

```

~~~~~ A$=STRING$(30,229)

■■■■■■■■■■ A$=CHR$(239)+CHR$(224)

■-■-■-■-■-■-■-■-■-■- A$=CHR$(239)+CHR$(241)

▲▲▲▲▲▲▲▲▲▲▲▲▲▲▲ A$=CHR$(252)+CHR$(254)

▲■■■■■■■■■■ A$=CHR$(239)+CHR$(252)+CHR$(254)

```

In text mode, use FOR X=1 TO N: LPRINT CHR\$(30);A\$; NEXT X

GRAPHIC MODEBIT-IMAGE CODE SEQUENCES

```

◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆ A$=CHR$(136)+CHR$(156)+CHR$(190)+CHR$(255)+
CHR$(190)+CHR$(156)+CHR$(136)+CHR$(128)

~~~~~ A$=CHR$(156)+CHR$(184)+CHR$(240)+CHR$(184)+
CHR$(156)+CHR$(142)+CHR$(135)

◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆ A$=CHR$(136)+CHR$(156)+STRING$(2,190)+STRING$(2,
255)+STRING$(2,190)+CHR$(156);CHR$(136)

+++++ A$=STRING$(2,136)+CHR$(190)+STRING$(2,136)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx A$=STRING$(2,179)+STRING$(2,140)

```

In graphic mode, use FOR X=1 TO N: LPRINT CHR\$(18);A\$; NEXT X

FIGURE 6-3. Simple borders produced on the DMP-400 by repeating concatenated codes for built-in block-graphic characters in text mode (upper set) and for bit-image dot codes in graphic mode (lower set).

When it comes to fancier borders—the so-called decorative borders—only the bit-image approach is feasible because of the resolution required. But this will often call for more than a single printhead pass, and figuring out the code sequences for many of the more baroque or gingerbread styles may take too much time or too many passes to consider doing them on the printer.

The computer and printer have ways of making up for this shortcoming, though. It is a simple matter for the computer to select a few dot patterns at random and tell the printer to print the sequence of selected codes in different ways. To get a random selection of dot codes stored in RAM, a FOR-

NEXT loop enclosing the statement $A(I) = \text{RND}(128) - 1$ for seven-pin printers, or $A(I) = \text{RND}(256) - 1$ for eight-pin printers, will do the job.

Once the codes are stored in the A(I) array, they can be printed in forward and backward orders, which amounts to printing an image and then its horizontal mirror image immediately afterward. Then the image-and-mirror-image pattern can be repeated many times in a single pass of the printhead. This will give you a border design, but how decorative it will be depends on the luck of the draw in the code selections by the computer.

PROGRAM 6-1. DECBORDS. Decorative borders from randomly selected dot codes.

```

10 'PROGRAM 6-1: "DECBORDS" -- RANDOMLY SELECTED DOT PATTERNS FOR BORDERS
15 'FOR TRS-80 PRINTERS AND TRS-80 MODEL III (USE BOOTHE'S PRINTER DRIVER)
20 INPUT "LENGTH OF DOT-PATTERN SEQUENCE (UP TO 16)";Q
30 INPUT "NUMBER OF BORDERS TO BE PRINTED";N
40 LPRINT CHR$(30);CHR$(27);CHR$(31);          'BOLDFACE PRINTING
50 LPRINT CHR$(27);CHR$(23);                    'COMPRESSED DENSITY
60 DIM A(16)                                     'ARRAY SIZE (UP TO 16)
100 FOR K=1 TO N
110   FOR I=1 TO Q: A(I)=RND(128)-1: NEXT I      'RANDOM SELECTION, ARRAY STORAGE
120   FOR I=1 TO Q: PRINT A(I);: NEXT I         'COMPUTER-SCREEN PRINT OF CODES
130   FOR J=1 TO 20
140     FOR I=1 TO Q: LPRINT CHR$(18);CHR$(128+A(I));: NEXT I 'PRINT PATTERN
150     FOR I=Q TO 1 STEP -1: LPRINT CHR$(A(I)+128);: NEXT I 'REVERSE ORDER
160   NEXT J
170   LPRINT CHR$(30);CHR$(13);                 'LINE FEED (1/6 INCH)
180   FOR I=1 TO Q: LPRINT A(I);: NEXT I        'PRINT OF DOT CODES UNDER BORDER
190   LPRINT CHR$(13);                          'CARRIAGE RETURN, LINE FEED
200 NEXT K
210 END

```

In program 6-1, called DECBORDS for “decorative borders,” the user can order any number (N) of borders to be printed having a dot sequence of length Q by answering the INPUT prompts in lines 20 and 30. With a fifteen-inch printer in elite pitch, the length of the dot-code sequence is limited to sixteen in this program unless the number of repetitions of the sequence in forward and reverse orders (lines 140 and 150) is reduced from twenty in line 130. An actual (and typical) printout of several borders produced in succession on the DMP-400 is shown in figure 6-4.

Most of these designs, frankly, don't measure up to the

```

50 72 34 4 8 24 84 9
28 113 40 54 88 68 89 1
77 126 15 81 15 107 30 13
103 112 50 9 109 46 82 87
119 121 69 112 118 61 90 72
42 55 13 55 55 21 16 72
25 118 42 106 0 66 5 123
127 112 78 55 40 30 63 4
126 15 3 38 77 14 124 69
95 40 106 96 18 74 110 27
80 64 119 107 96 20 126 36
36 74 54 107 38 126 10 124

```

FIGURE 6-4. Use of randomly selected dot codes for producing borders in a single printhead pass (DECBORDS program).

decorative borders you see in the catalogues, although a few are somewhat interesting. If you set N high enough, say $N=1000$, you're bound to get some prettier ones, and some pretty ugly ones, too. With a few thousand printouts, you might come up with a work of genius, but if you do, let the computer take some of the credit!

And be sure to save the code numbers printed under the design so you can duplicate it anytime you want. You can change the DECBORDS program slightly to make reproductions—in place of `A(I)=RND(128)-1`: at line 110, put `READ A(I)`: and then just list the work-of-genius numbers in a DATA statement at the end of the program.

There's a quicker way to achieve your artistic triumph—use mirror-image codes. DECBORDS provides only a horizontal mirror image by printing the code sequence forward and backward. With borders made in two passes of the print-head, you could print the original set of dot codes that is

selected by RND in the first pass (forward and backward as before) and print the vertical mirror image of that set in the second pass (again backward and forward). This is what MIR-ACLES ("mirror-image spectacles," program 6-2) does.

PROGRAM 6-2. MIRACLES/TRS. Decorative borders and patterns using mirror images (TRS-80).

```

10 'PROGRAM 6-2: "MIRACLES/TRS" -- 7-PIN DECORATIVE BORDERS & PATTERNS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 PRINTERS AND TRS-80 MODEL III (USE BOOTHE'S PRINTER DRIVER)
30 LPRINT CHR$(30);CHR$(27);CHR$(31);          'BOLD PRINTING
40 LPRINT CHR$(27);CHR$(23);                    'ELITE DENSITY
50 INPUT "LENGTH OF DOT-PATTERN SEQUENCE (UP TO 16)";Q
60 INPUT "NUMBER OF VERTICAL REPETITIONS";R
70 INPUT "NUMBER OF PATTERNS TO BE PRINTED";N
80 DIM A(16),B(16)
90 FOR K=1 TO N
100  FOR I=1 TO Q: RANDOM: A(I)=RND(128)-1: GOSUB 1000: NEXT I  'ARRAY STORAGE
110  FOR I=1 TO Q: PRINT A(I);: NEXT I: PRINT          'DISPLAY SELECTED CODES
120  FOR I=1 TO Q: PRINT B(I);: NEXT I                'DISPLAY MIRROR-IMAGE CODES
130  C=0
140  FOR J=1 TO 20
150    FOR I=1 TO Q: LPRINT CHR$(18);CHR$(128+A(I));: NEXT I  'PRINT PATTERN
160    FOR I=Q TO 1 STEP -1: LPRINT CHR$(A(I)+128);: NEXT I  'REVERSE ORDER
170  NEXT J
180  LPRINT CHR$(18);CHR$(13);                        'GRAPHIC (7/72") LINE FEED
190  FOR J=1 TO 20
200    FOR I=1 TO Q: LPRINT CHR$(128+B(I));: NEXT I  'PRINT MIRROR PATTERN
210    FOR I=Q TO 1 STEP -1: LPRINT CHR$(128+B(I));: NEXT I  'REVERSE ORDER
220  NEXT J
230  C=C+1: IF C<=R THEN LPRINT CHR$(18);CHR$(13);: GOTO 140
240  LPRINT CHR$(30);CHR$(13);                        '1/6" LINE FEED
250  FOR I=1 TO Q: LPRINT A(I);: NEXT I                'PRINT SELECTED DOT-CODES
260  LPRINT CHR$(13);
270  FOR I=1 TO Q: LPRINT B(I);: NEXT I                'PRINT MIRROR-IMAGE CODES
280  LPRINT CHR$(13);
290 NEXT K
300 END
1000 '***** MIRROR-IMAGE CONVERSION OF 7-BIT DOT-CODES *****
1010 '(MODIFIED VERSION OF PROGRAM 4-3, 'TRSTOEPS')
1020 NTRS=A(I)                                          'ENTER TRS-80 CODE, A(I)
1030 FOR P=6 TO 0 STEP -1                              '(7 PINS, NUMBERED 6 TO 0)
1040  X(P)=NTRS-2^P                                     '(CONVERSION OF
1050  IF X(P)<0 THEN Y$(P)="0": GOTO 1070              ' DECIMAL CODE
1060  Y$(P)="1": NTRS=X(P)                             ' TO BINARY FORM)
1070 NEXT P
1080 BYTE$=Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0)  'ORIGINAL BYTE (BINARY)
1090 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6) 'TURN BYTE UPSIDE DOWN
1100 B(I)=64*VAL(Y$(0))+32*VAL(Y$(1))+16*VAL(Y$(2))+8*VAL(Y$(3))+4*VAL(Y$(4))+2*
    VAL(Y$(5))+VAL(Y$(6))                             'CONVERT MIRROR BYTE TO DECIMAL
1110 RETURN

```

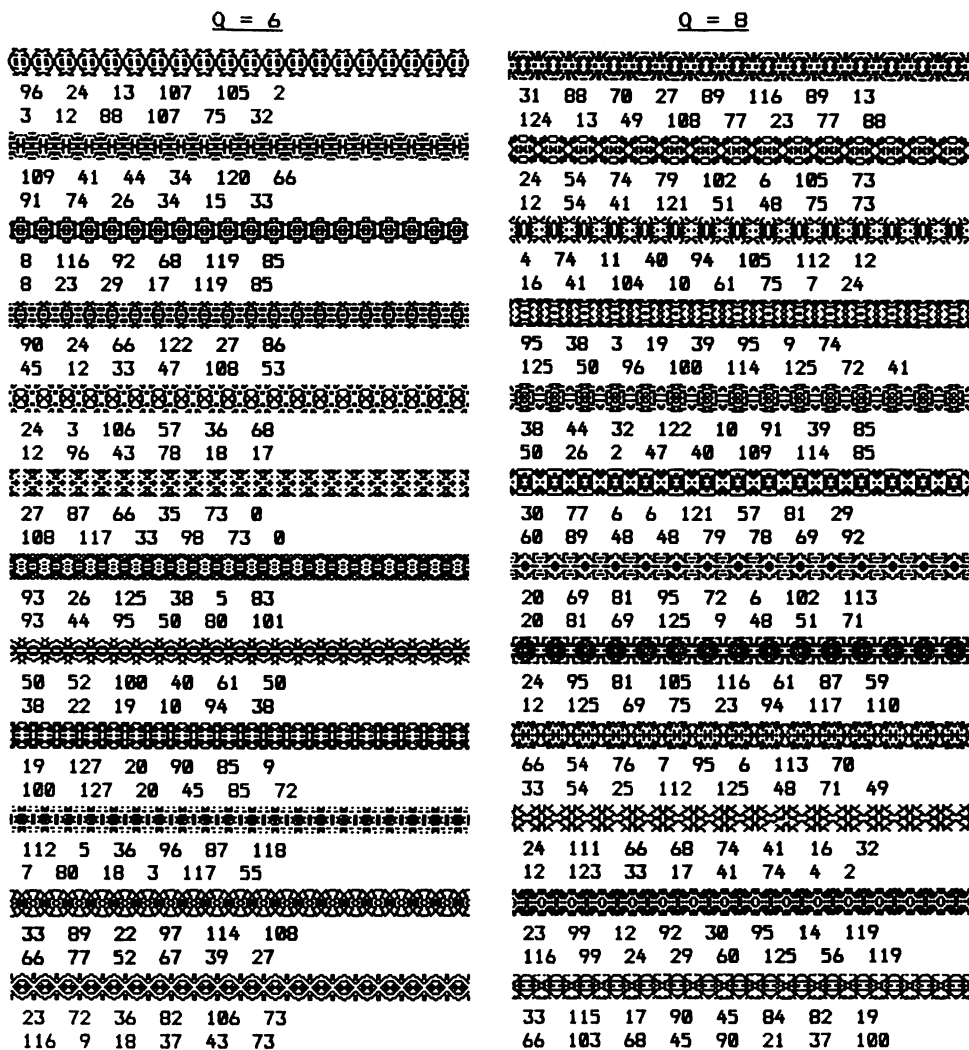


FIGURE 6-5. Using mirror-image dot codes to produce decorative borders by random selection (MIRACLES program with R=0). These borders are printed in two printhead passes, with the second pass being a vertical mirror image of the first. The randomly selected dot codes are displayed in the two rows of numbers under each border. The patterns on the left resulted from selections of six dot codes at a time, those on the right from eight-code selections.

Figure 6-5 shows what happens. The effect of adding the mirror-image line is really rather amazing and certainly says something about how much symmetry matters in visual beauty. With the vertical as well as horizontal symmetry provided by MIRACLES, we get nice-looking borders from over half—not just one out of every ten or so—of our random dot-code selections. And this could go on for thousands of print-outs!

The samples in figure 6-5 were produced by the DMP-400 using program 6-2, and thus were limited to seven-dot codes, or fourteen-dot heights in each two-pass border. There's room for more variation with eight-dot printers, so versions of MIRACLES for IBM-Epson and Apple printers are also presented (programs 6-3 and 6-4).

How do we get the mirror-image codes for these MIRACLES programs? From subroutines, such as the one called in line 100 of the DMP-400's program. These subroutines can be close copies of the EPSTOAPL (eight-bit) and TRSTOEPS (seven-bit) conversion programs in chapter 4 (programs 4-2 and 4-3).

PROGRAM 6-3. MIRACLES/EPS. Decorative borders and patterns (IBM-Epson).

```

10 'PROGRAM 6-3: "MIRACLES/EPS" -- 8-PIN DECORATIVE BORDERS AND PATTERNS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR THE EPSON FX-80 PRINTER DRIVEN BY TRS-80 MODEL 4 (USES '^' SYMBOL)
30 LPRINT CHR$(27)"@"; 'MASTER RESET
40 INPUT "LENGTH OF DOT-PATTERN SEQUENCE (UP TO 16)";Q
50 INPUT "NUMBER OF VERTICAL REPETITIONS";R
60 INPUT "NUMBER OF PATTERNS TO BE PRINTED";N
70 INPUT "PRINT DENSITY (0 TO 6)";D
80 DIM A(18),B(18)
90 FOR K=1 TO N
100 FOR I=1 TO Q: RANDOM: A(I)=RND(256)-1: GOSUB 1000: NEXT I 'ARRAY STORAGE
110 FOR I=1 TO Q: PRINT A(I);: NEXT I 'DISPLAY SELECTED CODES
120 FOR I=1 TO Q: PRINT B(I);: NEXT I 'DISPLAY MIRROR-IMAGE CODES
130 C=0 'START REPETITION COUNTER
140 G2%=FIX(40*Q)/256: G1%=40*Q-256*G2% 'DOT-COLUMNS NEEDED
150 GR$=CHR$(27)+"*"+CHR$(D)+CHR$(G1%)+CHR$(G2%) 'GRAPHIC RESERVATION
160 LPRINT GR$; 'RESERVE COLUMNS, DENSITY
170 FOR J=1 TO 20
180 FOR I=1 TO Q: LPRINT CHR$(A(I));: NEXT I 'PRINT SELECTED PATTERN
190 FOR I=Q TO 1 STEP -1: LPRINT CHR$(A(I));: NEXT I 'REVERSE ORDER
200 NEXT J
210 LPRINT CHR$(27)"A"CHR$(8)CHR$(10); '8/72" LINE FEED
220 LPRINT GR$;
230 FOR J=1 TO 20
240 FOR I=1 TO Q: LPRINT CHR$(B(I));: NEXT I 'PRINT MIRROR-IMAGE PATT.
250 FOR I=Q TO 1 STEP -1: LPRINT CHR$(B(I));: NEXT I 'REVERSE ORDER
260 NEXT J
270 C=C+1: IF C<=R THEN LPRINT CHR$(27)"A"CHR$(8)CHR$(13);: GOTO 160
280 LPRINT CHR$(27)"2"CHR$(13);
290 FOR I=1 TO Q: LPRINT A(I);: NEXT I 'PRINT SELECTED DOT-CODES
300 LPRINT CHR$(27)"2"CHR$(13);
310 FOR I=1 TO Q: LPRINT B(I);: NEXT I 'PRINT MIRROR-IMAGE DOT-CODES
320 LPRINT CHR$(27)"2"CHR$(13);
330 NEXT K
340 END

```



```

1000 '***** MIRROR-IMAGE CONVERSION OF 8-BIT DOT-CODES *****
1010 '(MODIFIED VERSION OF PROGRAM 4-2, 'EPSTOAPL')
1020 NEPS=A(I)                                'ENTER EPSON DOT-CODE, A(I)
1030 FOR P=7 TO 0 STEP -1                      '(8 PINS, NUMBERED 7 TO 0)
1040   X(P)=NEPS-2[P]                          '(CONVERSION OF DECIMAL
1050   IF X(P)<0 THEN Y$(P)="0": GOTO 1070      'MAL CODE TO
1060   Y$(P)="1": NEPS=X(P)                    'BINARY FORM)
1070 NEXT P
1080 BYTE$=Y$(7)+Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0) 'ORIGINAL
1090 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6)+Y$(7) 'MIRROR IMAGE
1100 NAPL=128*VAL(Y$(0))+64*VAL(Y$(1))+32*VAL(Y$(2))+16*VAL(Y$(3))+8*VAL(Y$(4))+
    4*VAL(Y$(5))+2*VAL(Y$(6))+VAL(Y$(7))      'CONVERT MIRROR-BYTE TO DECIMAL
1110 B(I)=NAPL                                'TAG CONVERTED CODE AS B(I)
1120 RETURN

```

PROGRAM 6-4. MIRACLES/APL. Decorative borders and patterns (Apple).

```

10 REM -- PROGRAM 6-4: "MIRACLES/APL" -- BORDERS AND PATTERNS
15 REM -- Copyright (c) 1985 by John Warner Davenport
20 REM -- FOR APPLE IIe COMPUTERS WITH APPLESOFT BASIC AND APPLE IMAGEWRITER
30 INPUT "LENGTH OF SEQUENCE ";Q
40 INPUT "NUMBER OF VERTICAL REPETITIONS ";R
50 INPUT "NUMBER OF PATTERNS TO BE PRINTED ";N
100 FOR K=1 TO N
110   PR#0
120   FOR I=1 TO Q
130     A(I)=INT(255*RND(256))
140     GOSUB 1000
150   NEXT I
160   FOR I=1 TO Q: PRINT A(I);" ";: NEXT I: PRINT
170   FOR I=1 TO Q: PRINT B(I);" ";: NEXT I: PRINT
180   C=0
190   PR#1
200   PRINT CHR$(24);: REM -- CLEAR PRINTER'S BUFFER
210   PRINT CHR$(27)+"E";: REM -- PRINT DENSITY = 96 DPI
220   GOSUB 2000
230   FOR J=1 TO 40
240     FOR I=1 TO Q
250       IF PEEK(49305) <> 16 THEN GOTO 250
260       POKE 49304,A(I)
270     NEXT I
280     FOR I=Q TO 1 STEP -1
290       IF PEEK(49305) <> 16 THEN GOTO 290
300       POKE 49304,A(I)
310     NEXT I
320   NEXT J
330   PRINT CHR$(27)+"T16"+CHR$(10);
340   GOSUB 2000
350   FOR J=1 TO 40
360     FOR I=1 TO Q
370       IF PEEK(49305) <> 16 THEN GOTO 370
380       POKE 49304,B(I)
390     NEXT I

```

```

400   FOR I=Q TO 1 STEP -1
410     IF PEEK(49305) <> 16 THEN GOTO 410
420     POKE 49304,B(I)
430     NEXT I
440   NEXT J
450   C=C+1: IF C=<R THEN PRINT CHR$(27)+"T16"+CHR$(10);: GOTO 220
460   PRINT CHR$(27)+"T24"+CHR$(10);
470   FOR I=1 TO Q: PRINT A(I);" ";: NEXT I
480   PRINT CHR$(27)+"T24"+CHR$(10);
490   FOR I=1 TO Q: PRINT B(I);" ";: NEXT I
500   PRINT CHR$(27)+"T32"+CHR$(10);
510 NEXT K
520 END
1000 REM -- ***** SUBROUTINE FOR MIRROR-IMAGE CONVERSION *****
1010 REM -- MODIFICATION OF "EPSTOAPL" (PROGRAM 4-2)
1020 NAPL=A(I)
1030 FOR P=7 TO 0 STEP -1
1040   X(P)=NAPL-2^P
1050   IF X(P)<0 THEN Y$(P)-"0": GOTO 1070
1060   Y$(P)="1": NAPL=X(P)
1070 NEXT P
1080 BYTE$=Y$(7)+Y$(6)+Y$(5)+Y$(4)+Y$(3)+Y$(2)+Y$(1)+Y$(0)
1090 MBYTE$=Y$(0)+Y$(1)+Y$(2)+Y$(3)+Y$(4)+Y$(5)+Y$(6)+Y$(7)
1100 NEPS=128*VAL(Y$(0))+64*VAL(Y$(1))+32*VAL(Y$(2))+16*VAL(Y$(3))+8*VAL(Y$(4))+
    4*VAL(Y$(5))+2*VAL(Y$(6))*VAL(Y$(7))
1110 B(I)=NEPS
1120 RETURN
2000 REM -- ***** SUBROUTINE FOR GRAPHIC RESERVATION *****
2010 RES=80*Q: L=LEN(STR$(RES))
2020 PRINT CHR$(27)+"G";
2030 FOR Z=1 TO 4-L: PRINT "0": NEXT Z
2040 PRINT RIGHT$(STR$(RES),L);
2050 RETURN

```

The processing time for generating mirror-image codes with these conversion programs is on the slow side, however, so you might consider speeding things up by replacing these subroutines with one consisting of a simple listing of the direct conversions, such as (with seven-bit codes):

```

1000 IF A(I)=0 THEN B(I)=0: RETURN
1001 IF A(I)=1 THEN B(I)=64: RETURN
1002 IF A(I)=2 THEN B(I)=32: RETURN
.
.
.
1127 IF A(I)=127 THEN B(I)=127: RETURN

```

This works in a pretty snappy way, since as soon as an IF turns out to be true in the computer's scanning of the list, its RETURN will send the program right back to array storage

and printing. With eight-bit printers, this method of doing the mirror-image conversions (the numbers for the conversions are in the first two figures of chapter 4) means typing 256 statements instead of the three dozen or so in EPSTOAPL.

It turns out that, whether you have a seven- or eight-dot program, the chances of getting a handsome border design for a MIRACLES program vary quite a bit with the length of the random code selection (Q). With very short sequences (Q=2 to 4), there usually aren't enough different dot patterns to generate interesting or pretty designs. And with the longest sequences (Q above 12), the visual pattern tends to get too complex to yield something pleasing. A certain degree of simplicity, combined with enough ingredients (Q=5 to 12) to get hundreds of possible sequences, seems to be needed for producing a high percentage of attractive borders.

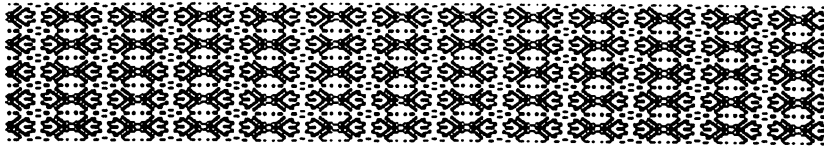
Screen, Tints, and Textures

Besides symmetry and simplicity, repetition is obviously adding a good deal to the pleasant effects of the more successful borders "designed" by the computer. So far we have had only horizontal repetition of image-and-mirror patterns in DECBORDS and MIRACLES, and we've been missing out on effects that could result from repeating the vertical image-and-mirror image. Built into the MIRACLES program is a feature to allow two-pass borders to keep repeating themselves before printing out the code numbers. If we use this feature, we'll get rhythm in another direction and a bigger bonus—new forms created by the butting of the bottom and top of each border design.

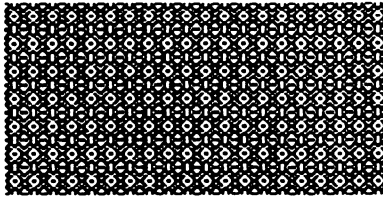
Whenever the prompt (NUMBER OF VERTICAL REPETITIONS?) is answered with a value of R above zero, the program will produce patterns that are too wide to be called borders, and in many ways more intriguing. Figure 6-6 presents some sample printouts of these multipass patterns.

Patterns produced by vertical repetitions are fascinating in their final printed form, and they are downright thrilling to watch as they are being printed. The peak of interest comes when the third line of printing starts, because it is usually difficult to tell from the initial two-pass border how the whole pattern is going to look. As soon as you see the third

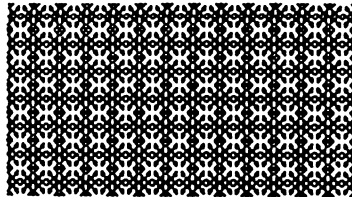
TRS-80 DMP-400



32 81 112 89 109 54 24 76 93 81 48 2
2 69 7 77 91 54 12 25 93 69 6 32

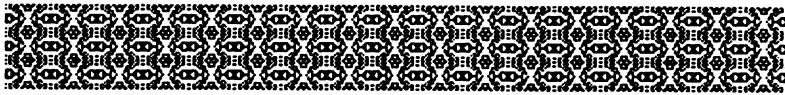


89 109 54 23 76
77 91 54 116 25

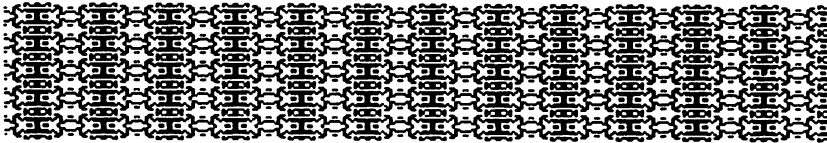


69 122 49 48 76 78 32
81 47 70 6 25 57 2

EPSON FX-80



167 6 228 167 83 233 249 87 156 198 113 208
229 96 39 229 202 151 159 234 57 99 142 11



20 16 32 60 70 10 16 11 89 90 26 123
20 4 2 30 49 40 4 104 77 45 44 111

FIGURE 6-6. Patterns produced by the DMP-400 and FX-80 printers by repeating two-pass border designs vertically as well as horizontally (MIRACLES program with R>0).

and fourth lines, you can't help thinking "Navajo!" or "Persian!" or "Colonial!" or "Scotch plaid!" and wondering whether it is a piece of wallpaper, a rug, or a flannel shirt that you've created on the printer.

Whether one of these patterns reminds you of a Navajo blanket or a madras jacket can also be determined by the print density used. Generally, the condensed density doesn't allow enough white space to show for clear, good-looking patterns, but that may be the best density for simulating the

tight, intricate styles favored in New Delhi. The elite density is usually the best compromise between the needs for white space and fine resolution, but American Indian designs are more readily suggested by printing in condensed-elongated mode, and styles of the British Isles seem more on the mark in standard density.

With high densities, such as the Apple printers' 160-dots-per-inch density, you're likely to have a dire shortage of white space, so you may want to repeat each randomly selected code before storage in the forward and backward arrays. Better yet, you can adjust the random selection itself so that more 0 codes are selected: for TRS-80 printers, instead of `A(I)=RND(128)-1` use `A(I)=RND(256)-1: IF A(I)>127 THEN A(I)=0`. With Applesoft's RND function the equivalent would be `A(I)=INT(512*RND(1))-1: IF A(I)>255 THEN A(I)=0`.) This trick can be used to generate more white space (and very complex borders or patterns with high values of Q) in other printers as well.

As with DECBORDS, you can add READ and DATA statements to the MIRACLES program for making copies of any designs you want to select from the random productions. Notice also that you can take a pattern that comes *close* to a design you want, put the dot codes that are under its printout into a DATA line, and edit the sequence—change one or two of the numbers in the DATA statement to make it *exactly* the design you want.

For that matter, why not use the READ-and-DATA form of MIRACLES for drawing nondecorative patterns from scratch? Once you get acquainted with the dot patterns, you can make good guesses ahead of time about how simple combinations of them will look in mirror images and repetitions. An upside-down V shape, for instance, results from the sequence of binary codes 64, 32, 16, 8, 4, 2, 1, 1, 2, 4, 8, 16, 32, and 64. If you put that sequence in a DATA line, and answer the prompts with Q=14 and R=6, the printer will present you with the lattice at the top of figure 6-7.

It is just as easy to generate the pattern of squares and the brick pattern in the middle part of figure 6-7. The bottom pattern—rows and rows of circles—is a particularly useful one, as we've already seen in chapter 5, because it is handy for working out dot codes for drawings. And when you get involved in designing symbols, letters, and numbers, you

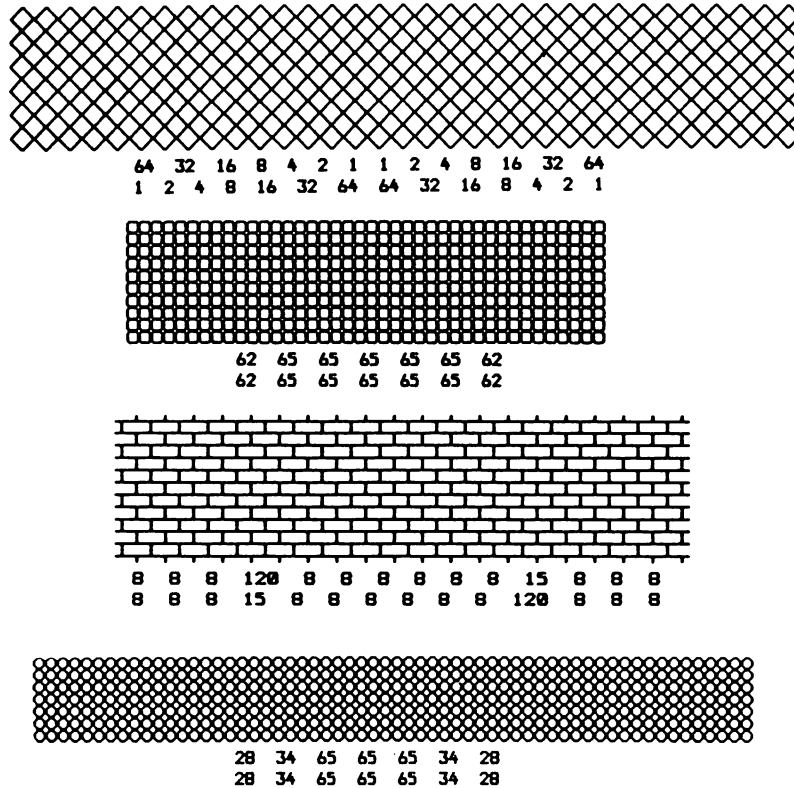


FIGURE 6-7. Printouts from an altered version of the MIRACLES program in which preplanned selections of dot codes are entered into DATA statements instead of having random selections.

may find that these rows of circles are better than graph paper for coding—you may want to make a dozen or more pages of them.

Quite a few of the grids and patterns used in architectural drafting, mapmaking, and geological illustration can be simulated by variations of the DECBORDS and MIRACLES programs. Failing that, drawing programs similar to LINE-DRAW (programs 5-10 to 5-12) may fill the bill.

Some dry-transfer materials shown in the catalogues are probably impossible to do—decently, at any rate—on most dot-matrix printers in personal computing use today. This seems to be the case with the most complex architectural symbol sheets and with sheets having tints and textures of certain types.

Wouldn't you know it: Tints made of *dots* are hardest of all! Dot tints (also called shading films and dot screens) usually range from 10% to 80% blackness and have twenty-five to eighty-five dot-lines per inch. These are precision-made by the graphics companies, and they vary the size and even the shape of the individual dots in ways that the DMP-400 or Epson FX-80 can barely approach. Some of the graduated dot-and-line tint sheets, which have grays of near-white continuously changing to near-black and vice versa, or oblique lines with slightly varying slopes, present a challenge not likely to be met by any of today's dot-matrix printers, even printers having twenty-four-pin printheads.

But there are a few useful things in this line that seven- or eight-pin printers can manage. With a program called SHADINGS (program 6-5), designed for IBM-Epson printers, we can make random-dot tints varying in small steps from white to black, and graduated tints as well.

The SHADINGS program is like the DECBORDS program in having sequences of dot codes selected randomly. It is even more like the shading subroutines in SPOTLITE (program 5-13) in chapter 5, and is in fact a direct extension of those. Instead of drawing randomly from the whole range of binary dot codes from 0 to 127 (as in DECBORDS), SHADINGS draws from much smaller, selected groups of dot codes at any one time (as in SPOTLITE).

In the ordered series of subroutines in the SHADINGS program, you can see several different ways of selecting codes at random. The codes given are as useful for Tandy or Apple printers as for IBM-Epson printers; no mirror-image conversions are needed. The code selections do not contain any common trouble codes.

PROGRAM 6-5. SHADINGS. Graduated dot tints.

```

10 'PROGRAM 6-5: "SHADINGS" -- 15 DIFFERENT SHADING DENSITIES
11 'FOR GRADUATED DOT TINTS OR SHADING RECTANGLES
12 'FOR IBM-EPSON PRINTERS, TRS-80 COMPUTERS
13 'Copyright (c) 1985 by John Warner Davenport
20 CLEAR 500: DEFINT Z,R,N,J,K,I,X,A
30 INPUT "GRADUATED (G) OR NOT (N)";GN$
40 IF GN$="N" THEN INPUT "SHADING VALUE (0 - 15)";Z
50 INPUT "NUMBER OF LINES PER SHADING";R
60 LPRINT CHR$(27)"a";CHR$(27)"3"CHR$(20); 'MASTER RESET, 20/216" LINE FEED SET
70 N=64
80 DIM A(N)

```

```

90 IF GN$="N" THEN GOTO 130
100 FOR J=1 TO 6: LPRINT CHR$(27)"L"CHR$(N)CHR$(0);STRING$(N,1);: NEXT J
    SOLID LINE ABOVE PURE-WHITE SPACE
110 LPRINT CHR$(13);
120 FOR Z=0 TO 15
130   FOR K=1 TO R
140     FOR I=1 TO N: RANDOM: X=RND(N): ON Z+1 GOSUB 1000,1100,1200,1300,1400,15
00,1600,1700,1800,1900,2000,2100,2200,2300,2400,2500: NEXT I
150     IF K=1 THEN PRINT: PRINT "SHADING VALUE = ";Z
160     FOR I=1 TO N: PRINT A(I);: NEXT I
170     FOR J=1 TO 3
175       LPRINT CHR$(27)"L"CHR$(2*N)CHR$(0);
180       FOR I=1 TO N: LPRINT CHR$(A(I));: NEXT I
190       FOR I=N TO 1 STEP -1: LPRINT CHR$(A(I));: NEXT I
200     NEXT J
210     LPRINT CHR$(13);
220   NEXT K
230   IF GN$="N" THEN GOTO 250
240 NEXT Z
250 END

998 REM -- SUBROUTINES FOR RANDOM SELECTIONS OF DOT CODES
999 REM -- (POSSIBLE TROUBLE CODES 9, 10, 12, 13, AND 112 NOT SELECTABLE)
1000 REM -- SHADING LEVEL 0:  BLANKS ONLY, 0% DENSITY
1001 A(I)=0: RETURN
1100 REM -- SHADING LEVEL 1:  7/448 DOT RATIO, 1.6% DENSITY
1101 IF X=1 THEN A(I)=1
1102 IF X=2 THEN A(I)=2
1103 IF X=3 THEN A(I)=4
1104 IF X=4 THEN A(I)=8
1105 IF X=5 THEN A(I)=16
1106 IF X=6 THEN A(I)=32
1107 IF X=7 THEN A(I)=64
1108 IF X>7 THEN A(I)=0
1109 RETURN
1200 REM -- SHADING LEVEL 2:  14/448 DOT RATIO, 3.1% DENSITY
1201 IF X>0 AND X<3 THEN A(I)=0
1202 IF X>2 AND X<5 THEN A(I)=1
1203 IF X>4 AND X<7 THEN A(I)=2
1204 IF X>6 AND X<9 THEN A(I)=4
1205 IF X>8 AND X<11 THEN A(I)=8
1206 IF X>10 AND X<13 THEN A(I)=16
1207 IF X>12 AND X<15 THEN A(I)=32
1208 IF X>14 AND X<17 THEN A(I)=64
1209 IF X>16 THEN A(I)=0
1210 RETURN
1300 REM -- SHADING LEVEL 3:  28/448 DOT RATIO, 6.2% DENSITY
1301 IF X>0 AND X<5 THEN A(I)=0
1302 IF X>4 AND X<9 THEN A(I)=1
1303 IF X>8 AND X<13 THEN A(I)=2
1304 IF X>12 AND X<17 THEN A(I)=4
1305 IF X>16 AND X<21 THEN A(I)=8
1306 IF X>20 AND X<25 THEN A(I)=16
1307 IF X>24 AND X<29 THEN A(I)=32
1308 IF X>28 AND X<33 THEN A(I)=64

```



```

1309 IF X>32 THEN A(I)=0
1310 RETURN
1400 REM -- SHADING LEVEL 4: 45/448 DOT RATIO, 10% DENSITY
1401 IF X>0 AND X<7 THEN A(I)=1
1402 IF X>6 AND X<13 THEN A(I)=2
1403 IF X>12 AND X<19 THEN A(I)=4
1404 IF X>18 AND X<25 THEN A(I)=8
1405 IF X>24 AND X<31 THEN A(I)=16
1406 IF X>30 AND X<37 THEN A(I)=32
1407 IF X>36 AND X<43 THEN A(I)=64
1408 IF X=43 THEN A(I)=1
1409 IF X=44 THEN A(I)=8
1410 IF X=45 THEN A(I)=64
1411 IF X>45 THEN A(I)=0
1412 RETURN
1500 REM -- SHADING LEVEL 5: 64/448 DOT RATIO, 14.3% DENSITY
1501 IF X>0 AND X<5 THEN A(I)=0
1502 IF X>4 AND X<9 THEN A(I)=1
1503 IF X>8 AND X<15 THEN A(I)=2
1504 IF X>14 AND X<21 THEN A(I)=4
1505 IF X>20 AND X<27 THEN A(I)=8
1506 IF X>26 AND X<33 THEN A(I)=16
1507 IF X>32 AND X<39 THEN A(I)=32
1508 IF X>38 AND X<43 THEN A(I)=64
1509 IF X>42 AND X<47 THEN A(I)=17
1510 IF X>46 AND X<51 THEN A(I)=66
1511 IF X>50 AND X<55 THEN A(I)=18
1512 IF X>54 AND X<59 THEN A(I)=40
1513 IF X>58 AND X<63 THEN A(I)=34
1514 IF X>52 THEN A(I)=1
1515 RETURN
1600 REM -- SHADING LEVEL 6: 96/448 DOT RATIO, 21.4% DENSITY
1601 IF INT(X/4)=<1 THEN A(I)=0
1602 IF INT(X/4)=2 THEN A(I)=1
1603 IF INT(X/4)=3 THEN A(I)=2
1604 IF INT(X/4)=4 THEN A(I)=4
1605 IF INT(X/4)=5 THEN A(I)=8
1606 IF INT(X/4)=6 THEN A(I)=16
1607 IF INT(X/4)=7 THEN A(I)=32
1608 IF INT(X/4)=8 THEN A(I)=64
1609 IF INT(X/4)=9 THEN A(I)=17
1610 IF INT(X/4)=10 THEN A(I)=33
1611 IF INT(X/4)=11 THEN A(I)=36
1612 IF INT(X/4)=12 THEN A(I)=40
1613 IF INT(X/4)=13 THEN A(I)=65
1614 IF INT(X/4)=14 THEN A(I)=66
1615 IF INT(X/4)=15 THEN A(I)=68
1616 IF INT(X/4)=16 THEN A(I)=18
1617 RETURN
1700 REM -- SHADING LEVEL 7: 136/448 DOT RATIO, 30% DENSITY
1701 X=INT(X/4): IF X=<1 THEN A(I)=5
1702 IF X=2 THEN A(I)=8
1703 IF X=3 THEN A(I)=24

```

```
1704 IF X=4 THEN A(I)=17
1705 IF X=5 THEN A(I)=18
1706 IF X=6 THEN A(I)=20
1707 IF X=7 THEN A(I)=33
1708 IF X=8 THEN A(I)=34
1709 IF X=9 THEN A(I)=36
1710 IF X=10 THEN A(I)=40
1711 IF X=11 THEN A(I)=68
1712 IF X=12 THEN A(I)=72
1713 IF X=13 THEN A(I)=80
1714 IF X=14 THEN A(I)=65
1715 IF X=15 THEN A(I)=81
1716 IF X=16 THEN A(I)=74
1717 RETURN
1800 REM -- SHADING LEVEL 8: 180/448 DOT RATIO, 40% DENSITY
1801 X=INT(X/4): IF X<1 THEN A(I)=11
1802 IF X=2 THEN A(I)=21
1803 IF X=3 THEN A(I)=25
1804 IF X=4 THEN A(I)=37
1805 IF X=5 THEN A(I)=41
1806 IF X=6 THEN A(I)=42
1807 IF X=7 THEN A(I)=49
1808 IF X=8 THEN A(I)=73
1809 IF X=9 THEN A(I)=81
1810 IF X=10 THEN A(I)=84
1811 IF X=11 THEN A(I)=88
1812 IF X=12 THEN A(I)=97
1813 IF X=13 THEN A(I)=100
1814 IF X=14 THEN A(I)=38
1815 IF X=15 THEN A(I)=34
1816 IF X=16 THEN A(I)=36
1817 RETURN
1900 REM -- SHADING LEVEL 9: 224/448 DOT RATIO, 50% DENSITY
1901 X=INT(X/4): IF X<1 THEN A(I)=85
1902 IF X=2 THEN A(I)=42
1903 IF X=3 THEN A(I)=82
1904 IF X=4 THEN A(I)=41
1905 IF X=5 THEN A(I)=37
1906 IF X=6 THEN A(I)=84
1907 IF X=7 THEN A(I)=101
1908 IF X=8 THEN A(I)=43
1909 IF X=9 THEN A(I)=81
1910 IF X=10 THEN A(I)=83
1911 IF X=11 THEN A(I)=85
1912 IF X=12 THEN A(I)=44
1913 IF X=13 THEN A(I)=45
1914 IF X=14 THEN A(I)=77
1915 IF X=15 THEN A(I)=70
1916 IF X=16 THEN A(I)=86
1917 RETURN
2000 REM -- SHADING LEVEL 10: 268/448 DOT RATIO, 60% DENSITY
2001 X=INT(X/4): IF X<1 THEN A(I)=15
2002 IF X=2 THEN A(I)=39
```

```

2003 IF X=3 THEN A(I)=45
2004 IF X=4 THEN A(I)=51
2005 IF X=5 THEN A(I)=57
2006 IF X=6 THEN A(I)=77
2007 IF X=7 THEN A(I)=86
2008 IF X=8 THEN A(I)=92
2009 IF X=9 THEN A(I)=101
2010 IF X=10 THEN A(I)=105
2011 IF X=11 THEN A(I)=108
2012 IF X=12 THEN A(I)=114
2013 IF X=13 THEN A(I)=116
2014 IF X=14 THEN A(I)=107
2015 IF X=15 THEN A(I)=110
2016 IF X=16 THEN A(I)=103
2017 RETURN
2100 REM -- SHADING LEVEL 11: 312/448 DOT RATIO, 70% DENSITY
2101 X=INT(X/4): IF X<1 THEN A(I)=31
2102 IF X=2 THEN A(I)=47
2103 IF X=3 THEN A(I)=55
2104 IF X=4 THEN A(I)=59
2105 IF X=5 THEN A(I)=61
2106 IF X=6 THEN A(I)=79
2107 IF X=7 THEN A(I)=87
2108 IF X=8 THEN A(I)=93
2109 IF X=9 THEN A(I)=107
2110 IF X=10 THEN A(I)=109
2111 IF X=11 THEN A(I)=115
2112 IF X=12 THEN A(I)=118
2113 IF X=13 THEN A(I)=121
2114 IF X=14 THEN A(I)=124
2115 IF X=15 THEN A(I)=43
2116 IF X=16 THEN A(I)=62
2117 RETURN
2200 REM -- SHADING LEVEL 12: 356/448 DOT RATIO, 80% DENSITY
2201 X=INT(X/4): IF X<1 THEN A(I)=126
2202 IF X=2 THEN A(I)=125
2203 IF X=3 THEN A(I)=123
2204 IF X=4 THEN A(I)=119
2205 IF X=5 THEN A(I)=111
2206 IF X=6 THEN A(I)=95
2207 IF X=7 THEN A(I)=63
2208 IF X=8 THEN A(I)=111
2209 IF X=9 THEN A(I)=125
2210 IF X=10 THEN A(I)=55
2211 IF X=11 THEN A(I)=59
2212 IF X=12 THEN A(I)=79
2213 IF X=13 THEN A(I)=91
2214 IF X=14 THEN A(I)=103
2215 IF X=15 THEN A(I)=117
2216 IF X=16 THEN A(I)=124
2217 RETURN
2300 REM -- SHADING LEVEL 13: 392/448 DOT RATIO, 87.5% DENSITY
2301 X=INT(X/4): IF X<1 OR X=9 THEN A(I)=126
2302 IF X=2 OR X=10 THEN A(I)=125

```

```

2303 IF X=3 OR X=11 THEN A(I)=123
2304 IF X=4 OR X=12 THEN A(I)=119
2305 IF X=5 OR X=13 THEN A(I)=111
2306 IF X=6 OR X=14 THEN A(I)=95
2307 IF X=7 OR X=15 THEN A(I)=63
2308 IF X=8 OR X=16 THEN A(I)=127
2309 RETURN
2400 REM -- SHADING LEVEL 14: 424/448 DOT RATIO, 95% DENSITY
2401 X=INT(X/4): IF X=<1 THEN A(I)=126
2402 IF X=2 THEN A(I)=125
2403 IF X=3 THEN A(I)=123
2404 IF X=4 OR X=8 THEN A(I)=127
2405 IF X=5 THEN A(I)=111
2406 IF X=6 THEN A(I)=95
2407 IF X=7 THEN A(I)=63
2408 IF X>8 AND X<17 THEN A(I)=127
2409 RETURN
2500 REM -- SHADING LEVEL 15: 448/448 DOT RATIO, 100% DENSITY
2501 A(I)=127: RETURN

```

Figure 6-8 shows fifteen levels of shadings resulting from random selections of dot codes from fourteen different groups of restricted-range codes (#1 through #14) and from simply repeating binary code 127 (#15, pure black). Instead of drawing from two to sixteen at a time as in DECBORDS, the SHADINGS program selects sixty-four dot codes in each sampling. And although each sequence is printed forward and backward a few times as in DECBORDS, the hope is that with a sequence as long as sixty-four, the repetition won't be very noticeable.

For the lightest degrees of shading (levels 1 to 4), the random selection is limited to single-dot codes (1, 2, 4, 8, 16, 32, and 64) and the blank code (0). At level 1, only 7 of the 64 numbers selected by RND will print a dot at all—that is $\frac{7}{448}$ or only 1.56% of the highest possible number of dots (448) that a seven-pin printhead could print in sixty-four prints with a full range of dot codes (0 to 127). Near the dark end of the shading series (levels 13 and 14), only codes printing six or seven dots are available for random selection. At value 15, the chances of a dot print are 98.44% (almost Ivory Soap in reverse) or 441 dots out of the possible 448.

The user of SHADINGS is offered a choice (line 30) between a printout of (a) a graduated tint in which each of the random-dot patterns shade into the next-darker shading level or (b) a particular degree of shading, again of the user's

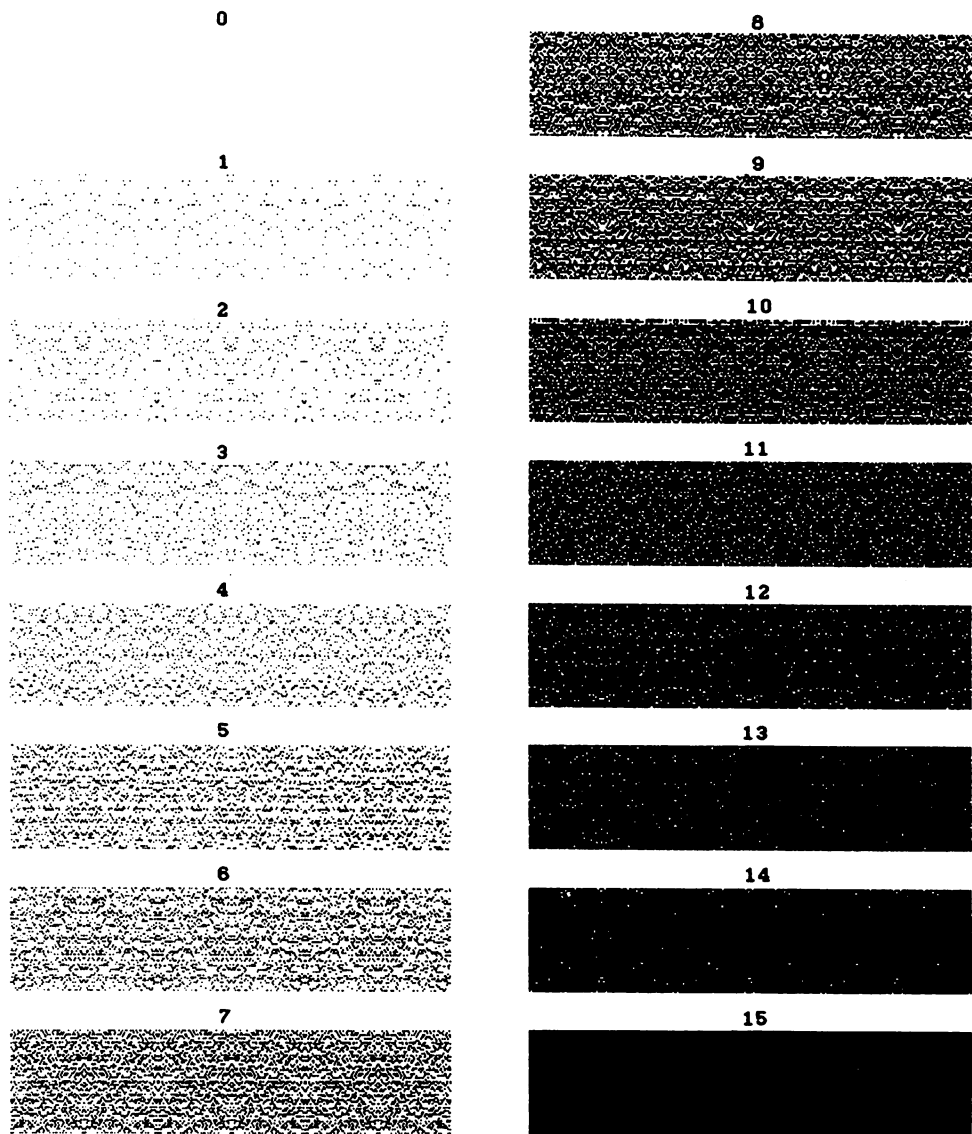


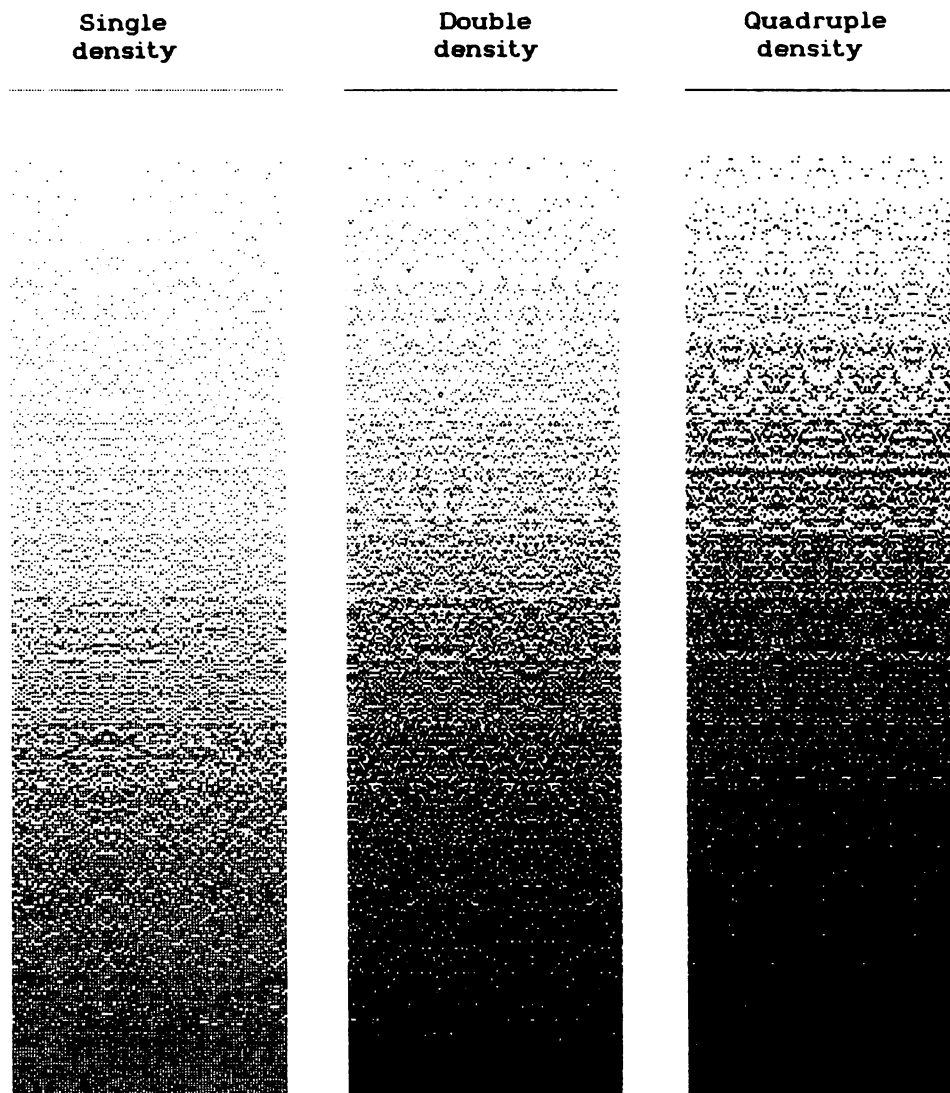
FIGURE 6-8. Fifteen densities of shading produced by random selections from restricted groups of dot codes in long sequences (SHADINGS program).

choice (line 40), with a selection of any of the sixteen darkness values in the 0-to-15 gamut. Regardless of whether (a) or (b) is chosen, a later INPUT prompt (line 50) lets the user specify how many lines (i.e., repetitions, R) of a given shading level are to be printed.

When you choose (a) and run the SHADINGS program with the FX-80's single, double, and quadruple print densities (by changing "L" to "K" or "Z" in lines 100 and 175),

you get graduated dot tints like those in figure 6-9. Notice how the print density affects the rate of change from light to dark shading. Also, if you study the printouts for the degree of repetition that is revealed in the dot-pattern sequences, you'll see that repeated forms are hardest to detect with single density and easiest with quadruple density. Changes from one shading level to the next are also easy to detect, in these printouts at every fourth print line (R was set to 4). If you try to make the graduation more continuous by dropping R to 1, then a whole printout will be less than two inches

FIGURE 6-9. Graduated dot tints produced by the SHADINGS program. Single (left), double (center), and quadruple (right) densities of the FX-80 were used.



high. The only way to get these tints to show the smooth transitions from light to dark like those in the catalogues, apparently, is to use many more shading levels—perhaps two or three times as many as in the SHADINGS program.

Graduated line tints are easier to program than graduated dot tints. In fact, the main idea in the programming of line tints has already been shown in chapter 5's ULTHIRES program (program 5-1)—varying the number of $\frac{1}{2}_{16}$ inch line feeds between horizontal lines. If you vary the number according to a sine function, you'll get printouts that resemble the line tints in the catalogues, provided your printer cooperates by being consistent in the size of its line feeds.

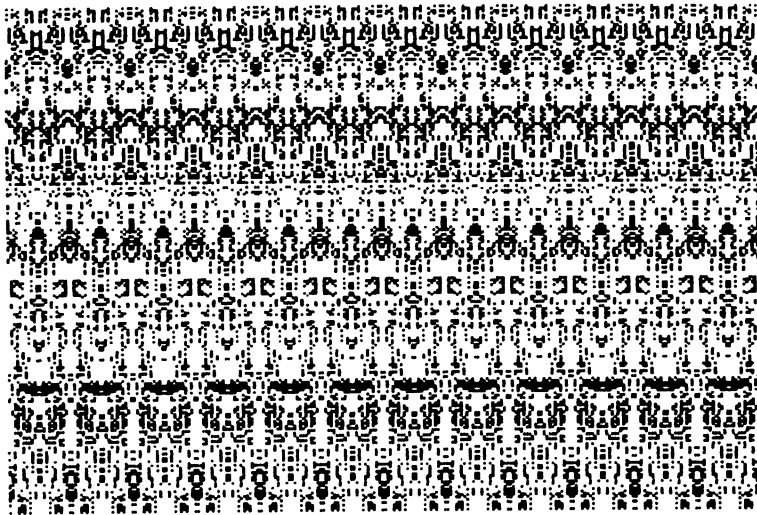
The SHADINGS program can be altered in several ways to generate special-effect textures. For instance, you can shorten the sequence of randomly selected dot codes and return to selecting from all 128 dot codes as in DECBORDS and MIRACLES. If you shorten the sequences to under 20 dot codes in length, you will no longer get random-looking dot patterns because the repetition of dot patterns becomes easy to detect, but this adds to their interest.

A good way to set up the sampling of all 128 codes is to delete lines 30, 40, 90–120, and 230 and change line 140 to **FOR I=1 TO N: A(I)=RND(LIMIT)-1: IF A(I)>127 THEN A(I)=0**. Because of the IF, **NEXT I** needs to be put in a new line (145). Normally LIMIT, which needs to be specified in a new line earlier, would be 128, but to increase the amount of white space you can set LIMIT to higher values. For shorter code sequences, you simply type in a lower value of N in line 70. And if you want print lines to be as long as they were before, increase the number of horizontal repetitions accordingly in line 170. For example, if you drop N to 16, change line 170 from **FOR J=1 TO 3** to **FOR J=1 TO 12**.

Depending again on the luck of the draw, this altered SHADINGS program will give you some sparkling tapestries that can often be even more fascinating than the patterns produced by vertical repetitions in the MIRACLES program. (Notice that instead of repeating the same two-pass symmetrical pattern many times, you are now generating a different dot pattern for each print line, while still getting the advantage of horizontal repetition with each line.) A typical printout is shown in figure 6-10's upper panel.

To add to the "special effects" of these textures, you can

WITHOUT BLACK-WHITE REVERSAL:



REVERSAL BETWEEN RANDOM SELECTIONS:

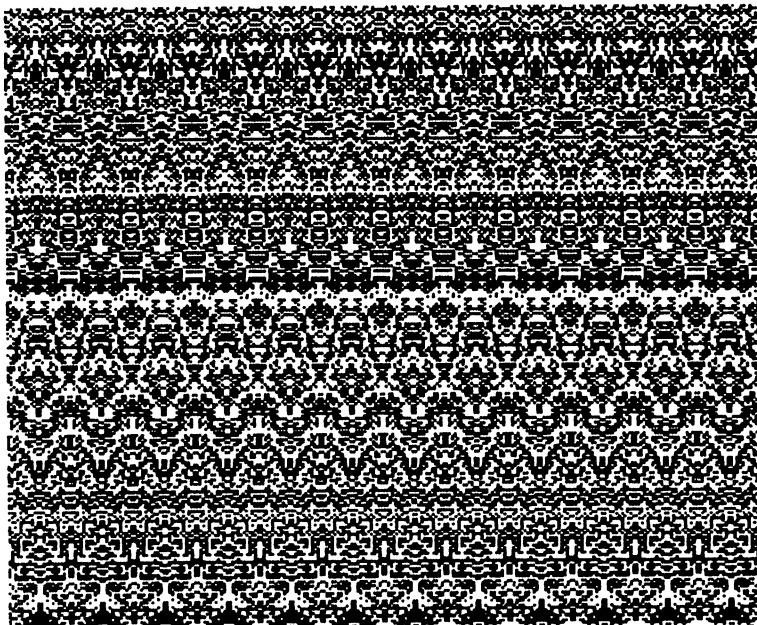


FIGURE 6-10. Special-effect textures produced on the DMP-400 by relatively short sequences of randomly selected dot codes. In the upper example, random selections followed each line feed and RND(200) was used instead of the usual RND(128). In the lower example, both positive and negative images were printed in separate print lines between random selections.

insert a black-to-white reversal of each line between the random selections. The lower panel of figure 6-10 shows one result of doing this. To provide the black-white reversal feature, add the following lines to the already altered SHADINGS program to provide the black-white reversal feature.

```

212 FOR J=1 TO 12
213 LPRINT CHR$(27)"L"CHR$(2*N)CHR$(0);
214 FOR I=1 TO N: LPRINT CHR$(127-A(I));:
      NEXT I
216 FOR I=N TO 1 STEP -1: LPRINT
      CHR$(127-A(I));: NEXT I
218 NEXT J: LPRINT CHR$(13);

```

By now we've changed the original SHADINGS program so much that it would be wise to consider the altered version a different program deserving a new name. As a suggestion, you can save it on tape or disk under the name TEXTURES.

Symbols

Some of the graphic symbols that appear on tapes and dry-transfer sheets are easy to do on the printer, and some are not. The fairly easy ones include those that make use of simple geometric forms such as squares, diamonds, and circles. Many types of arrows and pointers, register marks, traffic signs, and stars are also not too difficult. The same goes for the simpler symbols used in maps and architectural drawings. Coding sheets and programming approaches similar to those used for the cartoons in chapter 5 will be helpful in working out the codes. Two or three passes of the printhead may be all that are necessary for good-looking replicas of the catalogue symbols, assuming high-density printing is used.

Some two-pass symbols—ones more likely to be seen in the daily newspapers than in graphics catalogues—are illustrated in figure 6-11. This figure also shows that bit-image symbols can easily be intermixed with text characters; as soon as each spade, heart, diamond, or club symbol is printed in graphic mode, the program just shifts to text mode (for printing the numbers) on the same print line, provided semicolons after the graphic symbols are used to suppress line feeds.

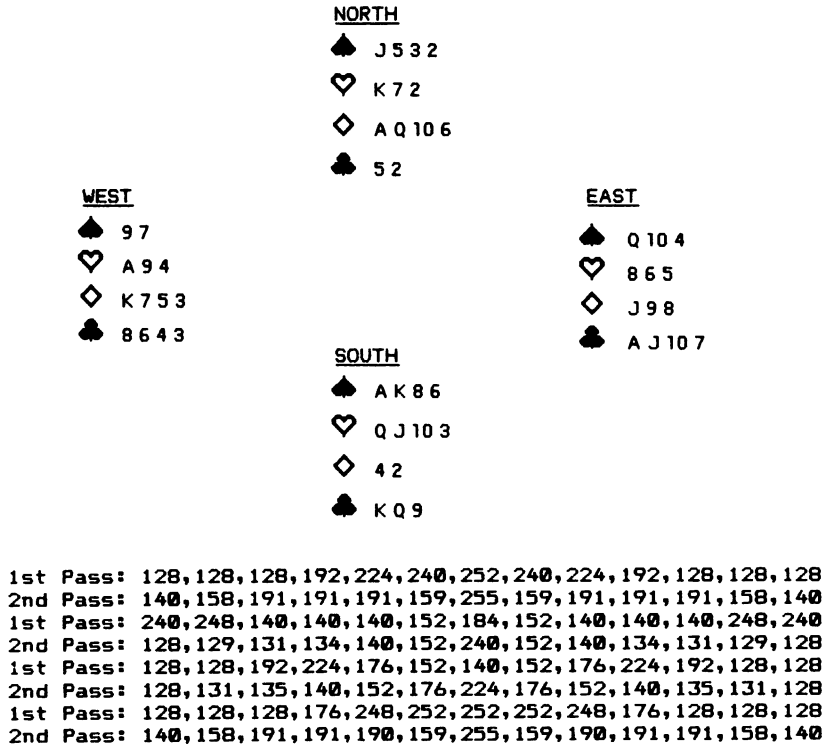


FIGURE 6-11. Bit-image bridge symbols combined with text-mode alphabetical characters.

Ultra-high-resolution symbols. Even with condensed printing, many of the more detailed symbols can't be handled very well using seven- or eight-dot printhead sweeps as in the LINEDRAW programs—the resolution is just not fine enough. What's needed is an ultra-high-resolution version of the LINEDRAW program—FINEDRAW (program 6-6).

PROGRAM 6-6. FINEDRAW. Drawing with $\frac{1}{216}$ -inch line feeds and a single pin of the printhead.

```

10 'PROGRAM 6-6: "FINEDRAW" -- ULTRA-HI-RES DRAWING PROGRAM (G CLEF SYMBOL)
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 CLEAR 1000
40 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18);
50 LF$=CHR$(27)+CHR$(51)
60 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
100 FOR R=1 TO 200
110 READ A: IF A=999 THEN GOTO 190
120 IF A=998 THEN GOTO 210

```

'CONDENSED PRINTING
'1/216" LINE FEED

```

130 IF A>0 THEN LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(A);: GOTO 110
140 READ M
150 IF BW$="W" THEN GOTO 170
160 LPRINT STRING$(-A,M+128);: GOTO 180
170 LPRINT STRING$(-A,136-M);
180 GOTO 110
190 LPRINT LF$;
200 NEXT R
210 END

1000 '*** POSITION GIVEN BY FIRST VALUE IN EACH DATA LINE ***
1001 DATA 120,-80,0,999
1002 DATA 120,-80,0,999
1003 DATA 120,-80,0,999
1004 DATA 120,-32,0,-5,8,-43,0,999
1005 DATA 120,-30,0,-9,8,-41,0,999
1006 DATA 120,-29,0,-12,8,-39,0,999
1007 DATA 120,-28,0,-3,8,-7,0,-4,8,-38,0,999
1008 DATA 120,-27,0,-3,8,-9,0,-4,8,-37,0,999
1009 DATA 120,-9,0,-2,8,-15,0,-3,8,-11,0,-4,8,-36,0,999
1010 DATA 120,-8,0,-4,8,-14,0,-2,8,-13,0,-3,8,-36,0,999
1011 DATA 120,-7,0,-6,8,-12,0,-2,8,-14,0,-4,8,-35,0,999
1012 DATA 120,-6,0,-8,8,-11,0,-1,8,-16,0,-3,8,-35,0,999
1013 DATA 120,-5,0,-1,8,-1,0,-8,8,-9,0,-1,8,-17,0,-3,8,-35,0,999
1014 DATA 120,-5,0,-1,8,-1,0,-8,8,-9,0,-1,8,-18,0,-3,8,-34,0,999
1015 DATA 120,-5,0,-1,8,-1,0,-8,8,-8,0,-1,8,-19,0,-3,8,-34,0,999
1016 DATA 120,-4,0,-1,8,-2,0,-8,8,-8,0,-1,8,-20,0,-2,8,-34,0,999
1017 DATA 120,-4,0,-1,8,-2,0,-8,8,-7,0,-1,8,-21,0,-3,8,-33,0,999
1018 DATA 120,-4,0,-1,8,-2,0,-8,8,-7,0,-1,8,-21,0,-3,8,-33,0,999
1019 DATA 120,-4,0,-1,8,-2,0,-8,8,-6,0,-1,8,-10,0,-3,8,-10,0,-2,8,-33,0,999
1020 DATA 120,-4,0,-1,8,-2,0,-8,8,-6,0,-1,8,-9,0,-6,8,-8,0,-3,8,-32,0,999
1021 DATA 120,-4,0,-1,8,-2,0,-8,8,-6,0,-1,8,-8,0,-8,8,-7,0,-3,8,-32,0,999
1022 DATA 120,-3,0,-1,8,-3,0,-8,8,-6,0,-1,8,-7,0,-2,8,-4,0,-3,8,-7,0,-3,8,-32,0,
    999
1023 DATA 120,-3,0,-1,8,-3,0,-8,8,-6,0,-1,8,-6,0,-2,8,-6,0,-3,8,-6,0,-4,8,-31,0,
    999
1024 DATA 120,-3,0,-1,8,-3,0,-8,8,-5,0,-1,8,-7,0,-1,8,-7,0,-3,8,-7,0,-3,8,-31,0,
    999
1025 DATA 120,-3,0,-1,8,-3,0,-8,8,-5,0,-1,8,-6,0,-1,8,-8,0,-3,8,-7,0,-3,8,-17,0,
    -6,8,-8,0,999
1026 DATA 120,-3,0,-1,8,-4,0,-6,8,-6,0,-1,8,-16,0,-3,8,-6,0,-3,8,-12,0,-11,8,-8,
    0,999
1027 DATA 120,-3,0,-1,8,-5,0,-4,8,-7,0,-1,8,-16,0,-3,8,-7,0,-3,8,-8,0,-5,8,-8,0,
    -2,8,-7,0,999
1028 DATA 120,-3,0,-1,8,-6,0,-2,8,-8,0,-1,8,-16,0,-3,8,-7,0,-3,8,-6,0,-3,8,-12,0,
    -2,8,-7,0,999
1029 DATA 120,-3,0,-1,8,-16,0,-1,8,-17,0,-2,8,-7,0,-3,8,-3,0,-3,8,-16,0,-2,8,-6,
    0,999
1030 DATA 120,-3,0,-1,8,-16,0,-1,8,-17,0,-2,8,-6,0,-7,8,-19,0,-2,8,-6,0,999
1031 DATA 120,-3,0,-1,8,-16,0,-1,8,-17,0,-2,8,-3,0,-4,8,-1,0,-3,8,-21,0,-3,8,-5,
    0,999
1032 DATA 120,-4,0,-1,8,-15,0,-1,8,-17,0,-5,8,-5,0,-3,8,-21,0,-3,8,-5,0,999
1033 DATA 120,-4,0,-1,8,-15,0,-1,8,-14,0,-5,8,-8,0,-4,8,-21,0,-3,8,-4,0,999
1034 DATA 120,-4,0,-1,8,-15,0,-1,8,-9,0,-5,8,-3,0,-2,8,-9,0,-3,8,-21,0,-3,8,-4,0,
    999

```

```

1035 DATA 120,-4,0,-1,8,-15,0,-1,8,-3,0,-6,8,-8,0,-2,8,-9,0,-4,8,-20,0,-3,8,-4,0
,999
1036 DATA 120,-5,0,-1,8,-14,0,-5,8,-13,0,-2,8,-9,0,-4,8,-20,0,-4,8,-3,0,999
1037 DATA 120,-6,0,-1,8,-8,0,-6,8,-17,0,-2,8,-10,0,-3,8,-20,0,-3,8,-4,0,999
1038 DATA 120,-7,0,-8,8,-5,0,-1,8,-17,0,-2,8,-10,0,-4,8,-19,0,-3,8,-4,0,999
1039 DATA 120,-20,0,-1,8,-16,0,-3,8,-10,0,-4,8,-19,0,-2,8,-5,0,999
1040 DATA 120,-20,0,-1,8,-16,0,-3,8,-11,0,-3,8,-19,0,-1,8,-6,0,999
1041 DATA 120,-20,0,-1,8,-16,0,-3,8,-11,0,-4,8,-18,0,-1,8,-6,0,999
1042 DATA 120,-20,0,-1,8,-16,0,-3,8,-12,0,-3,8,-17,0,-1,8,-7,0,999
1043 DATA 120,-20,0,-1,8,-15,0,-3,8,-13,0,-4,8,-15,0,-1,8,-8,0,999
1044 DATA 120,-20,0,-1,8,-15,0,-3,8,-14,0,-3,8,-14,0,-1,8,-9,0,999
1045 DATA 120,-20,0,-1,8,-14,0,-3,8,-16,0,-3,8,-12,0,-1,8,-10,0,999
1046 DATA 120,-21,0,-1,8,-13,0,-3,8,-17,0,-3,8,-10,0,-2,8,-10,0,999
1047 DATA 120,-21,0,-1,8,-12,0,-3,8,-19,0,-3,8,-8,0,-2,8,-11,0,999
1048 DATA 120,-21,0,-1,8,-11,0,-4,8,-20,0,-4,8,-4,0,-2,8,-13,0,999
1049 DATA 120,-21,0,-2,8,-10,0,-3,8,-23,0,-6,8,-15,0,999
1050 DATA 120,-21,0,-2,8,-9,0,-4,8,-44,0,999
1051 DATA 120,-21,0,-2,8,-8,0,-4,8,-45,0,999
1052 DATA 120,-22,0,-2,8,-5,0,-5,8,-46,0,999
1053 DATA 120,-22,0,-11,8,-47,0,999
1054 DATA 120,-22,0,-10,8,-48,0,999
1055 DATA 120,-23,0,-8,8,-49,0,999
1056 DATA 120,-25,0,-4,8,-51,0,999
1057 DATA 120,-80,0,999
1058 DATA 120,-80,0,999
1059 DATA 120,-80,0,999
1060 DATA 998

```

Whereas the lower-resolution LINEDRAW program uses the whole range of dot codes, FINEDRAW uses only one that prints anything—code 8 (middle dot)—and the blank code (0). And whereas LINEDRAW uses a standard line feed for graphic mode ($\frac{7}{32}$ inch with seven dots), FINEDRAW uses the tiny $\frac{1}{216}$ -inch line feed. This makes a huge difference in the number of printhead passes needed to draw a figure. If you recall that $\frac{1}{216}$ inch is only a third of a dot in size, you'll realize that for every single (seven-dot) sweep of the printhead using LINEDRAW, we're substituting twenty-one (single-dot) sweeps using FINEDRAW.

That means we'll often have twenty-one times as many DATA lines in FINEDRAW, too. But they'll be much simpler: Except for special codes, there are only the two dot codes (8 and 0) to worry about. Negative numbers are used in FINEDRAW as in LINEDRAW, except that we have a new rule: *Every* 8 or 0 must be preceded by a negative number, even when you don't want to repeat (in which case -1 is used, meaning "print the following code only once").

This rule was put in to give ourselves full freedom to use positive numbers for positioning (note line 130), line feeds (code 999 as before), or special subroutine calls. The latter are not illustrated in program 6-6, but these could take many forms, even a temporary shift from FINEDRAW to LINEDRAW.

This rule also makes it easier to see the simple alternation patterns that form in DATA statements when you follow the practice of starting a new DATA line for each new line of print. As lines 1013–1018 of program 6-6 show most clearly, you can see not only the alternation of 0 and 8 codes easily but also how the codes line up vertically when several DATA lines are listed on the screen or on the printer. This is a godsend for detecting and editing errors in data entry.

Suppose we try out the FINEDRAW program on a challenging symbol that has plenty of curves—the G clef. First we'll have to figure out the codes for each DATA and print line. Assuming condensed printing (the only density that makes sense with FINEDRAW) and also assuming that the printhead will travel parallel to the central shaft of the G clef, we'll have to start with a drawing like that on the left side of figure 6-12.

With some guesswork and trial and error, we draw a G clef that is elongated about ten units for every seven units in the final printed form, on ten-squares-per-inch graph paper or (as in figure 6-12) on seven-by-ten matrices of circles drawn by the printer (as in figure 6-7, bottom). So far, the process is just about the same as with the distorted drawing in WHITE-CAT (figure 5-21), except that we've gotten a little help from where the G clef lies in relation to the five lines of the musical staff (E-G-B-D-F, remember?).

At this point we could use the drawing to work out the codes for a LINEDRAW version of the G clef, and for comparison, we will. For a G clef drawn that way, we need only three passes of the printhead to cover the eighteen squares (which are eighteen dots) of the symbol's width. For FINEDRAW, however, these eighteen squares need to be expanded to three times as many, since there will be three $\frac{1}{2}$ 16-inch line feeds for each dot-unit of width.

Hence the fat (right-hand) drawing of figure 6-12, from which we can directly read off the sequence of code-8 dots and code-0 blanks needed for each DATA and print line. It's

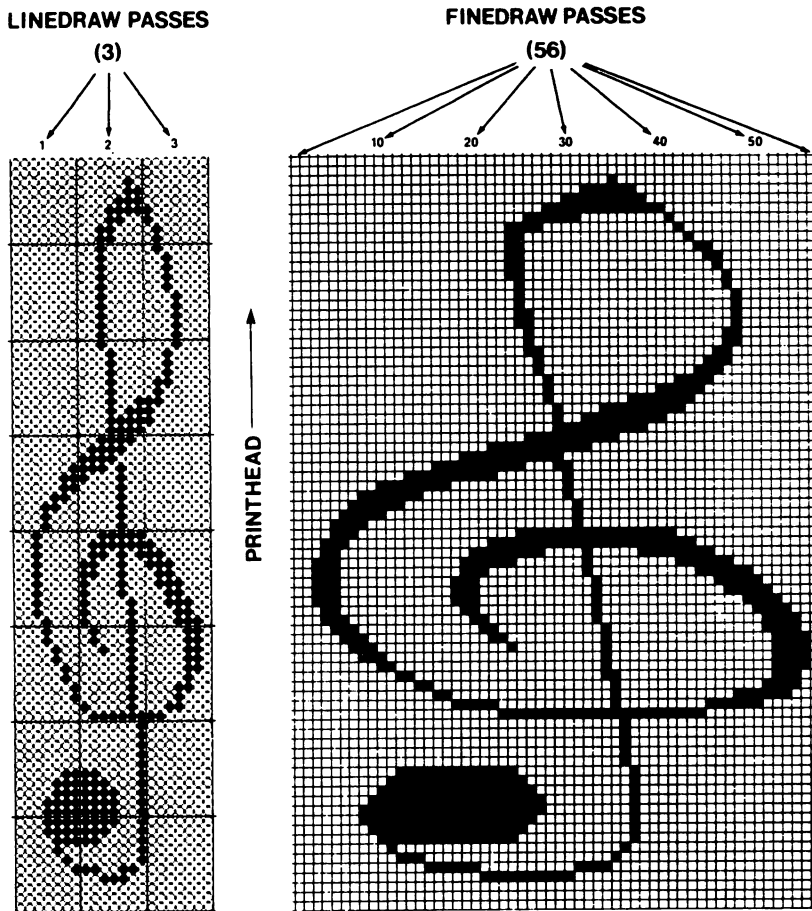


FIGURE 6-12. Coding sheet for working out dot codes for a FINEDRAW version of the G-clef symbol.

usually best to use the same numbering system for the DATA lines, print lines, and the lines in the “fat” drawing, to make editing easier.

Often the DATA values won’t come out exactly right the first time. When you get your first look at a printout you may think you can’t come any closer to perfection, but if you have a magnifying glass you’ll see the flaws easily. In the case of our G clef, our first-draft printout is shown on the left in figure 6-13. This doesn’t seem to need any more gilding of the lily, but if you “explode” the drawing, as in the right-hand portion of figure 6-13, by switching from condensed to pica density (or better yet, elongated-pica) and increasing the line feed (line 50), you’ll likely see some chances for further improvements (see the circled areas of the exploded G clef).

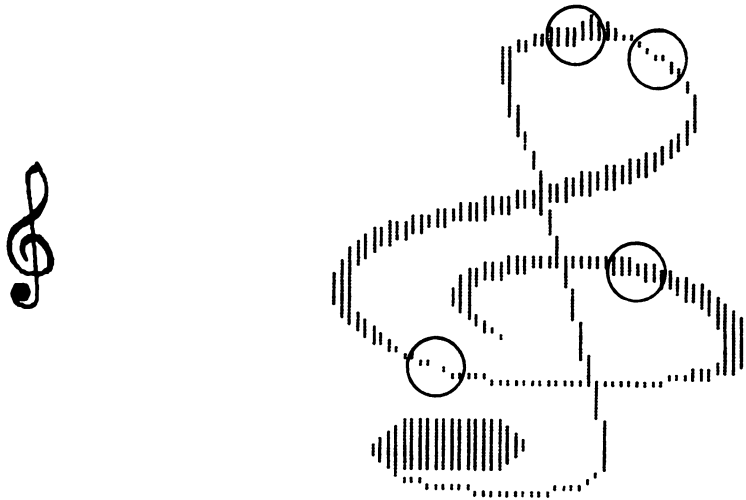


FIGURE 6-13. Left: printout of the FINEDRAW version of the G clef before final editing. Right: exploded view revealing imperfections (circled) to be corrected in the editing.

After editing our G clef, we got the comparison of LINEDRAW and FINEDRAW renditions shown in figure 6-14.

You can write versions of FINEDRAW for IBM-Epson and Apple printers pretty easily by noting the differences among the three versions of LINEDRAW (programs 5-10, 5-11, and 5-12) in chapter 5. For the Apple printers, the smallest line feed available— $\frac{1}{144}$ -inch—should produce nearly as good a printout as the printers having $\frac{1}{216}$ -inch feeds. If it doesn't, there is the consoling tradeoff that only two-thirds as many DATA and print lines would be needed.

Now that we've got the printer printing musical symbols, the obvious next step is to get the computer to listen to some real music, a live piano performance perhaps, and translate the sounds into bit-image codes for the printer. Publishers of

FIGURE 6-14. G clefs from the LINEDRAW program, printed in three printhead passes, and the FINEDRAW program, printed in fifty-three passes.

Drawn by LINEDRAW:

Drawn by FINEDRAW:



sheet music would no doubt find that handy, and so would composers. How to perform these marvels is beyond the scope of this book, but don't be surprised if the high-tech people have it all figured out by the time you're reading this. Even earlier than that, they will have reversed the process by getting a computer to read some printed music and play it through a synthesizer.

Logos. Life is too short to draw very many symbols the FINEDRAW way, certainly not pages of sheet music. But if you have only one or two figures to draw, it can make sense to invest this kind of effort. This is likely to be the case with decorative initials, special insignia, company logos, and other large symbols.

FINEDRAW is not the only way to make logos and letter-head designs, nor is LINEDRAW the only alternative. In the Epson FX-80 manual, Kater has three whole chapters on making logos and other symbols with user-defined characters that can be stored in that printer and retrieved with ASCII codes. Presumably the same sort of approach can be taken with the Apple Imagewriter and its "custom characters," and the manual for that printer also provides a detailed chapter on this topic.

If you're tempted to use these techniques or the LINEDRAW approach, remember that a good final product can be achieved with some of the old tricks of the commercial graphics business, such as making a figure extra large and then shrinking it to a final size in which flaws become invisible. Also, don't forget black-white reversals and mirror-image reversals, by BASIC or by camera.

We began this chapter by asking how closely the dot-matrix printer (meaning the most common eight- or nine-pin printers) can match the quality of the tapes and dry-transfer sheets of the graphics catalogues. The honest answer to this seems to be: surprisingly well on ordinary drafting tapes and decorative borders, also nicely on some complex symbols and logos (but with lots of work involved), less well (but often passably) on common symbols, and poorly on the most detailed screen tints and curvy symbols. If you're disappointed by this outcome, consider the efforts summarized in this chapter as merely a warm-up for some more serious work on drawing with eighteen- or twenty-four-pin printers.

7

Your Own Alphabets

MORE than 70% of the space in graphics catalogues is devoted not to borders, textures, and symbols but to letters and numbers. In dot-matrix printer graphics we have reason to be concerned about making our own letters and numbers, because the built-in characters of all printers fall short of meeting our needs in graphs, diagrams, and tables. At times these ready-made characters are too small, at other times too big. Often they're not facing the right way on the page, and sometimes they're difficult to line up precisely with hash marks on the axis of a graph. And there are times when only professional-looking typefaces will do the job.

So let's see what homemade alphabets we can build out of the bit-image dot patterns.

How to Make Numbers and Letters

The first thing we'll need is some graph paper, or better yet, some printouts of blank dot matrices such as the ones we produced in the last chapter (figure 6-7, bottom drawing), for figuring out the dot codes.

Coding sheets. Then we'll have to decide what size of characters we want and what direction we want them to face (orientation on the page) when they're printed. These things will depend a whole lot on whether we want to be able to print each character in a single sweep of the printhead. With

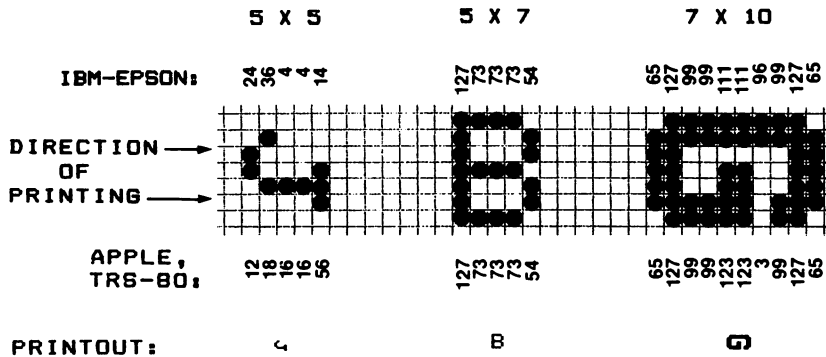


FIGURE 7-1. Coding sheet for homemade letters of different size and orientation.

a single pass of a seven-pin printhead, we won't be able to make numbers or letters much bigger than the built-in ones, but we do have some leeway about size and quite a bit of freedom to vary their style. Figure 7-1 shows three different letters we can get out of the seven-dot width of a single sweep.

In the left part of this figure you can see that the code sequence 24+36+4+4+14 on IBM-Epson printers (12+18+16+16+56 on Apple and Tandy printers) prints out as a tiny but legible J that is turned at a right angle from the direction the built-in characters face. In the center sample, we get a somewhat larger letter B facing still another direction from IBM-Epson sequence 127+73+73+73+54 and (by chance, because of vertical symmetry) the same sequence on Apple and Tandy printers. In the right-hand side of the figure we have a larger, bold-style G produced by the chain 65+127+99+99+111+111+99+99+127+65 on IBM-Epson printers and 65+127+99+99+123+123+3+99+127+65 on the Apple and Tandy printers (128 needs to be added for the Tandys, as usual).

To make a whole alphabet, you have to decide on the size of the dot matrix (five by seven, seven by ten, etc.) that will fit all the letters and numbers of that character set, and then draw the characters on the coding sheet. Then, regardless of the size of the characters, you have at least four choices as to their orientation: upright (right side up, like the built-in characters), perpendicular (turned ninety degrees to the right from the built-in characters), upside down (opposite of up-

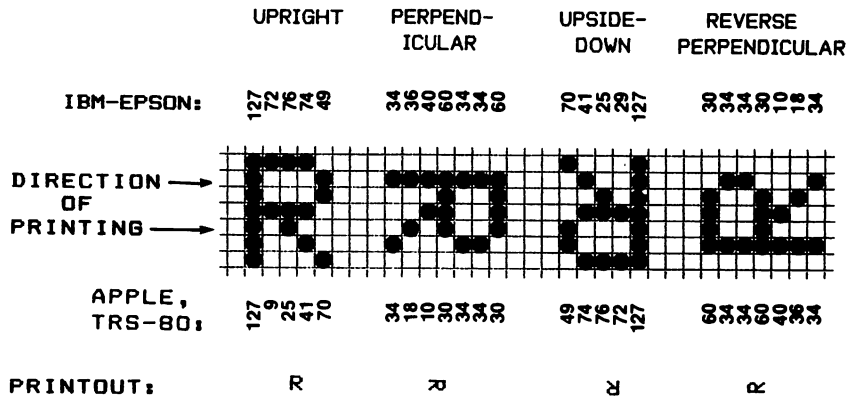


FIGURE 7-2. Coding sheet showing different dot-code sequences for the four different orientations of the same character.

right), and reverse perpendicular (opposite of perpendicular).

As figure 7-2 shows, the orientation will determine which particular dot codes are required for a given character as well as the order of the codes. Give this figure some extra study, because to understand it you have to visualize the movement of the printhead as going from the left margin of the paper toward the right side, and you also have to keep in mind where the printhead's upper dots (the #1, #2, and #4 pins on Apple and TRS-80 printers, the #64, #32, and #16 pins on IBM-Epson printers) are.

Most of the alphabets whose characters can be printed in a single printhead pass use small enough dot matrices to meet the requirements for "user-defined" characters with the Epson FX-80 or "custom characters" with the Apple Imagewriter. In both cases, there is a rather complicated series of special codes for defining, storing, and retrieving these homemade characters (see the manuals for details). With the FX-80, the characters (which can be symbols other than letters and numbers) must be small enough to fit into a matrix that is eight dots high and six columns wide, but you can consider this an eight-by-eleven matrix because (as with the built-in characters) the five intermediate positions between the six columns can be used, and either the top eight or the bottom eight of the FX-80's nine pins (but not all nine at a time) can be used.

With the Imagewriter, custom characters can also use

either the top or bottom eight of the nine pins, but unlike the FX-80, the characters can be up to sixteen dots wide. With both printers, you should note that the storage of homemade characters in the printer's RAM is temporary; if you shut off the power, they will be lost. So a format file that is stored on tape or disk will be needed for storing the dot-code sequences and "installing" these custom characters—and the special control codes for storing and retrieving them—every time you want to use them after booting up the system.

If your printer doesn't have the custom-character feature, there is no need to weep. All the storing and retrieving can be done by the computer and dot-code sequences can be sent to the printer for printing homemade characters nearly as quickly. Naturally, after putting in all the time to work out the dot codes for the characters, you would be storing this information on tape or disk anyway. And control codes as well as dot codes can be built into the definitions of the characters, as we'll see later in this chapter.

In the examples of character sets that follow, the printouts will be from the DMP-400 (usually) and the FX-80 (occasionally). Code sequences for the characters will not be presented in most cases, but full coding for the more important alphabets can be found in appendix B. The codes are in binary-sum form, so that they can also be used for Apple printers.

The smallest alphabet. A five-by-five matrix appears to be the smallest that will provide decent-looking, legible characters. When these are in the perpendicular orientation they can be made taller or shorter by varying the print density. The upper part of figure 7-3 displays some five-by-five perpendicular characters in standard and elongated sizes as they appear in DMP-400 printouts.

When printed at higher densities, these five-by-five characters are in many cases too small to be clearly legible, and for many purposes they may be too primitive in style, at whatever density, to be useful. In a tight squeeze, though, they'll help solve a problem.

Near-normal-sized alphabets. The lower part of figure 7-3 shows a homemade five-by-seven alphabet in several densities, and the (DMP-400) printer's built-in capital letters are

5 X 5 ALPHABETSTANDARD

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

CONDENSED-ELONGATED

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

5 X 7 ALPHABETBUILT-IN CHARACTERS:

<@UQWLEUIHJYJZORORST>3X>N

STANDARD

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

ELITE

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

CONDENSED

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

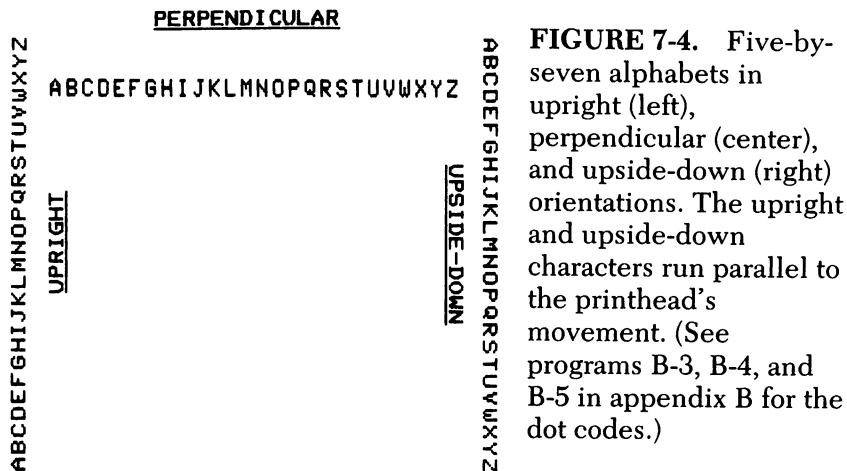
CONDENSED-ELONGATED

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

FIGURE 7-3. Perpendicular five-by-five and five-by-seven alphabets printed on the DMP-400. For the latter characters, a comparison with the printer's built-in characters is provided. (See programs B-2 and B-4 in appendix B for the dot codes.)

also shown for comparison. At the standard density of 60 dots per inch, these perpendicular five-by-sevens are slightly taller than the built-ins; the sizes are nearly identical when the homemade characters are printed in elite (72-dots-per-inch) density.

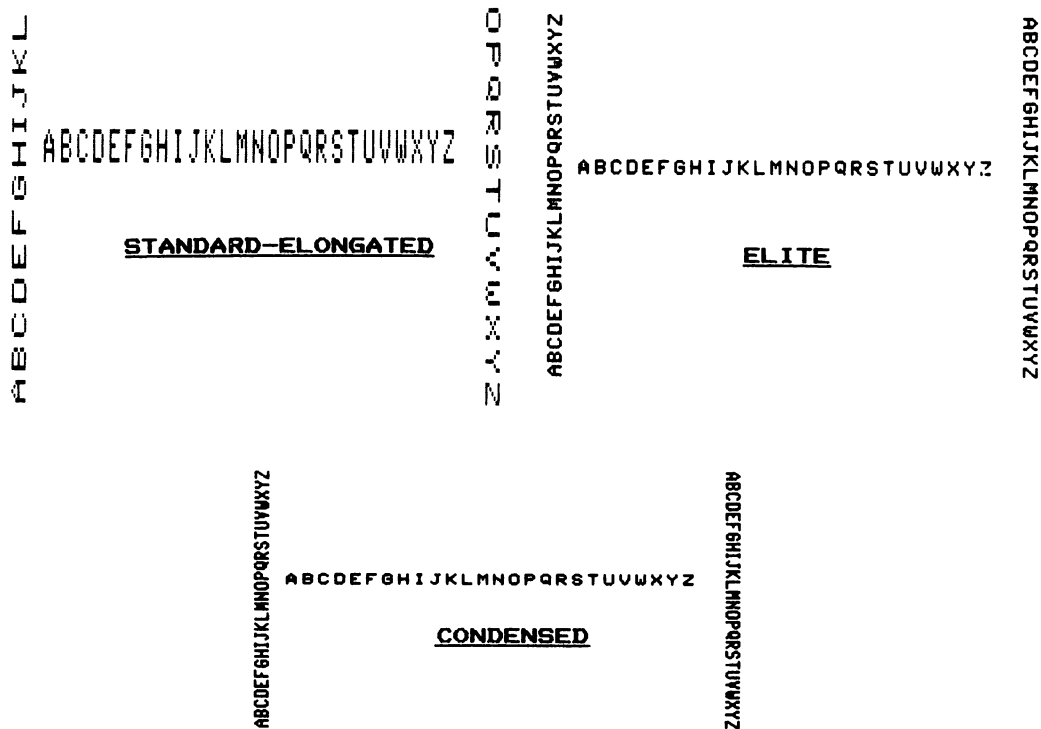
Figure 7-4 shows five-by-seven do-it-yourself character sets in upright, perpendicular, and upside-down orientations. Three different sets of dot-code sequences were nec-



essary for these. All three orientations are printed in standard density.

Compare these alphabet sets with those in figure 7-5, which are printed in elongated, elite, and condensed pitches. See how the uprights and upside downs get fatter and thinner, while the perpendiculars get taller or shorter, as the density is varied. For a view of reverse perpendiculars, turn the page upside down.

FIGURE 7-5. How upright and upside-down characters are affected differently from perpendiculars by changes in the DMP-400's printing density.



The DMP-400's five-by-seven perpendicular set has lower- as well as uppercase letters (see figure 7-3). The lowercase letters that have descenders ("legs")—letters g, p, q, and y—have backspace control codes as well as dot codes in their string definitions. That is, the dot-code sequence for each of these letters is preceded by the sequence `CHR$(30)+CHR$(8)+CHR$(4)+CHR$(18)` to start the printing two dot-spaces lower than for the other characters. This little trick is easy in the case of perpendiculars but of no use for descenders in the upright or upside-down alphabets.

Unless we are taking advantage of features such as the Epson FX-80's provisions for user-defined characters, we can't make our upright letters exactly like the built-in characters because we are restricted to whole dot-columns and not allowed to address intermediate positions between them that the printer's ready-made characters use. But we can approach the sharpness of the built-ins in a perpendicular alphabet if we use a high density and a five-by-eleven matrix. Figure 7-6 shows the five-by-eleven condensed alphabet that makes use of the denser packing of the dots in the DMP-400's condensed pitch.

If you're planning to use normal-sized homemade characters for camera-ready copy, you may run into difficulties in the photographic reproduction of five-by-seven characters

FIGURE 7-6. A perpendicular five-by-eleven condensed alphabet designed for 100-dots-per-inch density on the DMP-400. (See programs B-6 and B-7 in appendix B for the dot codes.)

5 X 11 CONDENSED ALPHABET

CONDENSED

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

ELITE

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

STANDARD

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

because of their loose dot packing. Much better results will show up with the five-by-eleven condensed letters. A comparable alphabet can be produced on IBM-Epson printers by using their double density (120 dots per inch) and a five-by-thirteen matrix. For the best quality on Apple printers use a five-by-seventeen matrix with the highest density (160 dots per inch). (The numbers thirteen and seventeen are chosen here instead of twelve and sixteen because they are odd numbers, which tend to work better than even numbers in designing alphabetical characters.)

Black-white reversals often don't come out clear and legible with any of the homemade alphabets we've seen so far. With the five-by-five and five-by-seven characters printed in standard or elite densities, you can't reduce their size photographically by more than about 25% before they become pretty hard to read—there's just not enough white area. The same holds for printing in the DMP-400's condensed pitch, the Epson double density, and the Imagewriter's only density without any photographic reduction. But with five-by-eleven, five-by-thirteen, or five-by-seventeen matrices you have a lot more room for maneuvering to get a good, legible white-on-black. For example, you can change two or three codes in each letter's code sequence so that the letters contain more white space.

We can also make our letters easier to read, and a bit more stylish, by adding serifs. If we make better use of the seven-dot width in a seven-by-eleven perpendicular set, for example, we can get a serified alphabet that is quite useful. Figure 7-7 shows what our seven-by-eleven serif characters look like on the DMP-400 along with some seven-by-sixteen serifs on the Epson FX-80.

TRS-80, Apple, and IBM users who envy those having italic alphabets in their printers' ROMs—such as the Epson FX-80 users—should especially take note of what can be achieved with homemade alphabets. Figure 7-8 shows that a fairly good italic in both upper- and lowercase can be managed with the DMP-400 in condensed pitch. This character set was designed in the upright orientation, the same way as the built-in italics on the FX-80 face, using a seven-by-nine matrix. The code sequences (appendix B), converted to Epson coding, should work for the IBM Graphics Printer in

7 X 11 SERIF ALPHABETSTANDARD

ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890112

ELITE

ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890112

CONDENSED

ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890112

7 X 16 SERIF ALPHABET (EPSON FX-80)

A B C D E F G H I J K L M

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

N O P Q R S T U V W X Y Z

FIGURE 7-7. Serif alphabets: seven-by-eleven perpendiculars from the DMP-40 and seven-by-sixteen perpendiculars from the Epson FX-80 (double density). (See programs B-10 and B-11 in appendix B for the dot codes.)

FIGURE 7-8. An upright italic character set for TRS-80 printers. (See program B-8 in appendix B for the dot codes.)

ABCDEFGHIJKLMNOPQRSTUVWXYZ! " # \$

*a b c d e f g h i j k l m n o p q r s t u v w x y z % & ' **

1 2 3 4 5 6 7 8 9 0 () + , - . / [] \ ^ _ '

Personal Computer Graphics

7 X 10 BOLD ALPHABETELITE-ELONGATED

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

CONDENSED-ELONGATED

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

STANDARD

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
abcdefghijklmnopqrstuvwxyz

ELITE

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
abcdefghijklmnopqrstuvwxyz

CONDENSED

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

FIGURE 7-9. Seven-by-ten perpendicular bold characters (DMP-400). (See program B-9 in appendix B for the dot codes.)

double density, although the characters would be slightly narrower.

The alphabets we've seen so far may be a bit too dainty for things like labeling a graph. But we still haven't run out of ways to print characters in a single printhead pass. Let's try a homemade boldface style.

Figures 7-9 and 7-10 show two bold alphabets, a seven-by-ten bold perpendicular for Radio Shack printers and an eight-by-eighteen bold perpendicular for IBM-Epson printers. These are probably more useful for legends on graphs, can

B X 18 BOLD ALPHABET (EPSON FX-80)**SINGLE DENSITY****T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 10****T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 10****DOUBLE DENSITY****T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 10****T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 10**

FIGURE 7-10. Eight-by-eighteen perpendicular bold characters (FX-80). (See program B-12 in appendix B for the dot codes.)

still be printed in a single pass, and work well in black-white reversals. The reversals, however, require a black surround (i.e., a code-255 black print or two before and after the print of each character), or else there'll be no division between the outer white parts of the characters and the white paper on which the black parts are printed, making the characters semi-invisible.

A nine-pin alphabet. Labeling on graphs that are to be photographically reduced by 70% or more is in danger of becoming illegible, especially if the characters used are bold or medium bold and near the size of built-in characters. The Epson FX-80 gives us a way of solving this problem with its unique nine-pin graphic mode. Using all nine pins, we can make bigger letters (but light style instead of bold) and still print them in a single pass.

By adding a ninth (bottom) pin that is addressable, this mode provides twice as many dot patterns—512 instead of 256—as well as added height to characters in the upright orientation. It's almost as easy to enter the nine-pin mode as it is to use regular eight-pin graphics: Use the command `LPRINT CHR$(27)CHR$(94)CHR$(D)CHR$(N1)CHR$(N2)`, where `D=0` is for single density and `D=1` is for dou-

ble density (N1 and N2 specify the number of dot columns reserved for graphics as usual).

The dot codes for the top eight pins remain the same, but here comes the catch: It takes a two-byte code to fire nine-pin dot patterns. This is where Kalinowski's double-hexadecimal code system may save the day. If it takes the two-byte sequence such as CHR\$(240)+CHR\$(128) to produce a single print of a dot pattern, how much shorter it is to use HG+E0 instead! Also, it turns out that the second of the two bytes for each print is always either 128 or 0 (E0 or A0 in double-hex), because that second byte represents either the firing or the nonfiring of the ninth pin in the printhead.

In appendix B, you'll find a double-density nine-by-thirteen upright alphabet for the FX-80's nine-pin mode that uses double-hex codes for every character. The usual form of the code sequence defining a letter (say, R) is

```
R=RL+E0+E0+HV+E0+E8+E0+E8+A0+E8+A0+E8+A0
+E8+A0+EC+A0+EE+A0+EB+A0+E9+E0+CG+E0+B0
+E0+STRING$(4,0)
```

where RL is shorthand for the graphic reservation (AR+CU+A1+AF+A0, or 27+94+1+15+0, for reserving fifteen dot columns—thirteen for the letter and two for the space between letters). Many codes are shorter than this one, since wherever 128s or 0s occur in runs, the STRING\$ function can be used for repeating. If this still seems like a lot of work, consider how long the definition of a nine-pin letter R would be in CHR\$() coding.

In double density the nine-by-thirteen upright alphabet looks like the one shown in figure 7-11.

ABCDEFGHI I JKLMNOPQRSTUVWXYZ 1234567890¼

FIGURE 7-11. A nine-by-thirteen upright alphabet using the FX-80's nine-pin graphic mode in double-density and double-hexadecimal codes. (See program B-13 in appendix B for the dot codes.)

Word-Spelling Programs

It's one thing to combine dot patterns into letters and numbers, but quite another to combine these characters into words, phrases, and multidigit numbers. BASIC programs for

spelling words or forming large numbers will have to take into account the orientation of the characters on the page, their size and spacing, how the user tells the computer what to spell, and other matters.

As usual, there is more than one approach possible. The first that comes to mind for forming a word begins by defining the characters of the alphabet in concatenated dot-code strings. The upright italic letter C, for instance, can be labeled C\$ and be defined for Radio Shack printers as follows:

```
C$=CHR$(176)+CHR$(204)+STRING$(2,194)+
  STRING$(2,193)+CHR$(161)+CHR$(131)+
  CHR$(128)
```

After we've done this for all the letters in the alphabet, we can print a series of them in any order we want by using a simple LPRINT command. Assuming we are already in graphic mode and have cleared enough string space (say, two thousand bytes), the command can be of this form:

<i>BASIC Command</i>	<i>Printout</i>
LPRINT S\$;P\$;E\$;L\$;L\$; I\$;N\$;G\$	SPELLING

or of this form:

LPRINT S\$+P\$+E\$+L\$+L\$+ I\$+N\$+G\$	SPELLING
--	-----------------

or even of this form:

LPRINT S\$P\$E\$L\$L\$I\$N\$G\$	SPELLING
---------------------------------	-----------------

but not, unfortunately, of this form:

LPRINT SPELLING	(none!)
-----------------	---------

We can, however, get rid of all those dollar signs and see what we're spelling more clearly by defining all our variables as string-type with a DEFSTR A-Z statement early in the program. If you do this, make sure that the DEFSTR statement comes *after* a CLEAR statement and that any numeric variables in the program are given type-declaration tags (% , ! , or # suffixes) in their naming.

After DEFSTR, the computer will treat A the same as A\$ and consider it to contain the same dot-code sequence. For lowercase letters (e.g., g), GL will mean the same as GL\$.

And if you make the change to DEFSTR after you've already typed the character string definitions with dollar signs on their names, don't bother to delete them—BASIC doesn't care about that little redundancy.

Now let's work our way back toward the simplest way of spelling again:

<i>BASIC Command</i>	<i>Printout</i>
LPRINT S+P+E+L+L+I+N+G	SPELLING
LPRINT S;P;E;L;L;I;N;G	SPELLING
LPRINT SPELLING	(none!)

Still no luck with regular spelling, but at least having the letters separated by semicolons or plus signs makes for easier reading than by dollar signs. (Plus signs are easier to type than semicolons with most keyboards, because you don't have to keep releasing the shift key after each capital letter.) Don't try to separate by using spaces; the computer will just ignore them. You also won't get anywhere, except further along a blank page, by putting the letters inside quotation marks; this will just produce line feeds when the printer is in graphic mode or printing of built-in letters rather than your homemade letters when it's in text mode.

If your character strings don't have spaces in them (at the end of each dot-code sequence), you will have to separate the printing of different letters by using another kind of space, such as a string named SP that is defined by the blank code, CHR\$(128). And for separating words, you'll need another space-string defined by several blanks, say, STRING\$(6,128).

With the opposite orientation, upside-down characters, the same ways of getting proper spelling and spacing apply except that the order of the characters in LPRINT statements has to be reversed. If you want the word VOLUME to be printed in this orientation (as we did on the right-hand side of the stock market graph in chapter 1), then use LPRINT E+M+U+L+O+V instead of LPRINT V+O+L+U+M+E.

Part of the trick here is always to imagine the printhead sweeping from the left side of the paper toward the right (even if your printhead prints while moving in both directions). If the last letter in the word is supposed to be the one nearest to the left margin, then it'll only come out that way by backward spelling in the LPRINT command.

PROGRAM 7-1. SPELLUPS. Spelling with upright and upside-down homemade alphabets.

```

10 'PROGRAM 7-1: "SPELLUPS" -- SPELLING OF UPRIGHT-ORIENTED WORDS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 CLEAR 1000
40 DEFSTR A-Z          'ALL VARIABLES ARE STRING-TYPE EXCEPT THOSE WITH '%'
50 SP=STRING$(2,128)   '2-DOT SPACE
60 LPRINT CHR$(18);    'ENTER GRAPHIC MODE
70 GOSUB 1000          'MEMORIZE CHARACTER STRINGS
80 GOSUB 2000          'READ AND PRINT CHARACTERS
90 END

1000 REM -- UPRIGHT 5 X 7 LETTERS (CAPITALS)
1001 A=CHR$(252)+CHR$(138)+CHR$(137)+CHR$(138)+CHR$(252)+SP
1002 B=CHR$(255)+STRING$(3,201)+CHR$(182)+SP
1003 C=CHR$(190)+STRING$(3,193)+CHR$(162)+SP
1004 D=CHR$(255)+STRING$(3,193)+CHR$(190)+SP
1005 E=CHR$(255)+STRING$(2,201)+STRING$(2,193)+SP
1006 F=CHR$(255)+STRING$(2,137)+STRING$(2,129)+SP
1007 G=CHR$(190)+CHR$(193)+STRING$(2,201)+CHR$(186)+SP
1008 H=CHR$(255)+STRING$(3,136)+CHR$(255)+SP
1009 I=CHR$(193)+CHR$(255)+CHR$(193)+SP
1010 J=CHR$(160)+CHR$(192)+CHR$(193)+CHR$(191)+CHR$(129)+SP
1011 K=CHR$(255)+CHR$(136)+CHR$(148)+CHR$(162)+CHR$(193)+SP
1012 L=CHR$(255)+STRING$(4,192)+SP
1013 M=CHR$(255)+CHR$(130)+CHR$(156)+CHR$(130)+CHR$(255)+SP
1014 N=CHR$(255)+CHR$(132)+CHR$(136)+CHR$(144)+CHR$(255)+SP
1015 O=CHR$(190)+STRING$(3,193)+CHR$(190)+SP
1016 P=CHR$(255)+STRING$(3,137)+CHR$(134)+SP
1017 Q=CHR$(158)+CHR$(161)+CHR$(177)+CHR$(161)+CHR$(222)+SP
1018 R=CHR$(255)+CHR$(137)+CHR$(153)+CHR$(169)+CHR$(198)+SP
1019 S=CHR$(166)+STRING$(3,201)+CHR$(178)+SP
1020 T=STRING$(2,129)+CHR$(255)+STRING$(2,129)+SP
1021 U=CHR$(191)+STRING$(3,192)+CHR$(191)+SP
1022 V=CHR$(131)+CHR$(156)+CHR$(224)+CHR$(156)+CHR$(131)+SP
1023 W=CHR$(191)+CHR$(192)+CHR$(184)+CHR$(192)+CHR$(191)+SP
1024 X=CHR$(227)+CHR$(148)+CHR$(136)+CHR$(148)+CHR$(227)+SP
1025 Y=CHR$(131)+CHR$(132)+CHR$(248)+CHR$(132)+CHR$(131)+SP
1026 Z=CHR$(225)+CHR$(209)+CHR$(201)+CHR$(197)+CHR$(195)+SP
1027 HY=STRING$(3,136)+SP      'HYPHEN
1028 N4=CHR$(144)+CHR$(152)+CHR$(148)+CHR$(255)+CHR$(144)+SP
1029 RETURN

2000 READ ZZ: IF ZZ="END" THEN RETURN
2001 IF ZZ="A" THEN LPRINT A;
2002 IF ZZ="B" THEN LPRINT B;
2003 IF ZZ="C" THEN LPRINT C;
2004 IF ZZ="D" THEN LPRINT D;
2005 IF ZZ="E" THEN LPRINT E;
2006 IF ZZ="F" THEN LPRINT F;
2007 IF ZZ="G" THEN LPRINT G;
2008 IF ZZ="H" THEN LPRINT H;
2009 IF ZZ="I" THEN LPRINT I;
2010 IF ZZ="J" THEN LPRINT J;

```

```

2011 IF ZZ="K" THEN LPRINT K;
2012 IF ZZ="L" THEN LPRINT L;
2013 IF ZZ="M" THEN LPRINT M;
2014 IF ZZ="N" THEN LPRINT N;
2015 IF ZZ="O" THEN LPRINT O;
2016 IF ZZ="P" THEN LPRINT P;
2017 IF ZZ="Q" THEN LPRINT Q;
2018 IF ZZ="R" THEN LPRINT R;
2019 IF ZZ="S" THEN LPRINT S;
2020 IF ZZ="T" THEN LPRINT T;
2021 IF ZZ="U" THEN LPRINT U;
2022 IF ZZ="V" THEN LPRINT V;
2023 IF ZZ="W" THEN LPRINT W;
2024 IF ZZ="X" THEN LPRINT X;
2025 IF ZZ="Y" THEN LPRINT Y;
2026 IF ZZ="Z" THEN LPRINT Z;
2027 IF ZZ="-" THEN LPRINT HY;
2028 IF VAL(ZZ)>9 THEN P%=VAL(ZZ): GOSUB 6000: LPRINT PN;
2029 IF ZZ="" THEN LPRINT SP;SP;SP;SP;
2030 IF ZZ="*" THEN LPRINT LF;
2031 IF VAL(ZZ)=4 THEN LPRINT N4;
2032 IF ZZ="PERSONAL" THEN LPRINT P;E;R;S;O;N;A;L;
2033 IF ZZ="COMPUTER" THEN LPRINT C;O;M;P;U;T;E;R;
2034 IF ZZ="GRAPHICS" THEN LPRINT G;R;A;P;H;I;C;S;
2035 GOTO 2000
6000 '***** POSITIONING AND LINE FEED STRINGS *****
6010 QP%=FIX(P%/256): PP%=P%-256*QP%
6020 PN=CHR$(27)+CHR$(16)+CHR$(QP%)+CHR$(PP%)
6030 LF=CHR$(30)+CHR$(13)+CHR$(18)+PN
6040 RETURN
10000 '***** DATA *****
10010 DATA 60,D,O,T,-,M,A,T,R,I,X,,P,R,I,N,T,E,R,,D,M,P,-,4,0,0,*
10020 DATA 105,PERSONAL,,COMPUTER,,GRAPHICS,*
10030 DATA END

```

A spelling program for upright words. Regardless of whether you plan to print words right side up or upside down, you can beef things up by using spelling programs with READ and DATA statements. Program 7-1, called SPELLUPS (for “spelling of upright and upside-down characters”), shows how not only spelling but a few other things can be done this way.

Besides the READ-DATA looping, the prominent feature of this program is the list of IF...THEN LPRINT statements immediately after the READ statement. These translate the simple items of the DATA lines into print commands that use the string names for all the alphabetic characters previously defined in a subroutine (subroutine 1000). That makes spelling even easier for the user, who can set the computer keyboard for lowercase and use capitals or small letters as

desired, separating the letters with commas instead of semicolons, plus signs, or dollar signs.

But since almost every variable in SPELLUPS is a string variable (note line 40's `DEFSTR A-Z`), a DATA item can be much longer than a single letter—up to 255 characters long, in fact. So if the appropriate IF...THEN LPRINT statements have been put into the program, the user can type in frequently used words, phrases, or even sentences as single DATA items and do the typing in the normal fashion, without putting commas between the letters. For an example of this, see the DATA in line 10020 and how that word sequence is translated into printing by lines 2032–2034.

Using READ and DATA also makes spacing and punctuation easier. You can use an actual space from the keyboard's spacebar, or no space at all, between two commas in a DATA statement (either way it will be treated as a single space in the statement `IF ZZ="" THEN LPRINT SP;SP;SP;SP;` in line 2029). And you can use punctuation marks normally, provided you have defined their string names in bit-image code sequences and put in an IF...THEN LPRINT statement for each. Notice the code and translation for a hyphen in the SPELLUPS program.

Beyond letters, numbers, and punctuation, SPELLUPS also codes certain DATA items for positioning and line feeds, as the FINEDRAW program (program 6-6) does. In lines 10010 and 10020, the numerical DATA items are position codes telling the printer where to start printing the characters that immediately follow. Since all DATA items are interpreted by READ ZZ to be string data, the translations of these codes (line 2028) have to include the VAL function for converting a string to a numeric variable; eventually, VAL (ZZ) gets converted into a pair of numerics (QP% and PP%) that specify the dot-column position in the printer's direct-positioning command (here taking the form $27 + 16 + QP\% + PP\%$).

With the addition of a line-feed code (LF, or it could be some otherwise unused punctuation mark such as the asterisk), you have the kind of control that you need for the lettering on a graph. (The entire SPELLUPS program could, with minor alterations, be a subroutine in a program for printing a graph.) You also gain the ability to spell and label with homemade characters that require more than one printhead pass

to print, and we'll see plenty of examples of such characters later on.

Spelling with perpendiculars. Spelling in the perpendicular orientation calls for some programming changes from what we've just seen with upright and upside-down characters. For one thing, now each letter in a word needs a line feed immediately after it, or else the next letter will be printed right on top of it. Also, since a line feed precedes nearly every letter printed, a position code will be needed to tell the printer at what dot column (or tab stop) to start printing—every letter! (This assumes that you are not trying to print at the left margin and that you are leaving the printer's DIP switches controlling line feeds and carriage returns in their normal settings.)

But the programming need not be very complex if you just want to print a few words with LPRINT statements and avoid typing all the IF...THEN LPRINT statements we have in the SPELLUPS program. One way to save a lot of headaches is to build the positioning and line-feed codes into the dot-code sequences defining the character strings of your alphabet. Start by defining all untagged variables as strings (with **DEFSTR A-Z** again) and further defining a line-feed code (for example, **LF=CHR\$(30)+CHR\$(27)+CHR\$(56)+CHR\$(10)** for the DMP-400's $\frac{9}{16}$ -inch line spacing) and an abbreviation code for positioning, such as **PS=CHR\$(27)+CHR\$(16)+CHR\$(QP%)+CHR\$(PP%)**, at an early point in your program. Then add PS+ to the front end of each dot-code sequence for a character and tack a +LF onto its tail end. Taking an example from the seven-by-ten bold perpendicular alphabet, change the coding for capital letter T from **T=STRING\$(8,152)+STRING\$(2,254)** to **T=PS+STRING\$(8,152)+STRING\$(2,254)+LF**.

With a subroutine for converting dot-column positions (P%) into QP% and PP% position codes, commands to print words or phrases can take the form **P%=(dot-column):GOSUB 1000:LPRINT W+O+R+D**. For proper positioning, putting the P% value and its conversion before the LPRINT will ensure that the numerical values of the positioning codes, QP% and PP%, will get into the PS code that each of the characters W, O, R, and D has at the beginning of its string sequence, and the LF at the end of each sequence will move each printed

letter out of the way for the next to be printed with proper spacing. The order of events here is critical, because if the values of QP% and PP% are not read into the PS code before the character's LPRINT command, both values will be interpreted as zero, meaning the printing of the character will start at the left margin.

For more elaborate programming of perpendicular spelling, see the WRDCRAFT listing (program 7-2). That program shows another way of getting the values of QP% and PP% into each character's PS code before that character is called upon to be printed. Like SPELLUPS, WRDCRAFT uses READ-and-DATA looping and a list of IF...THEN LPRINT translations. But instead of having a single positioning statement at the head of this list (as in SPELLUPS) or putting PS into character strings (as just suggested for occasional spelling of perpendiculars), WRDCRAFT has PS immediately following LPRINT in the IF...THEN LPRINT statement for each character. To provide the QP% and PP% values for all the PSs, the conversion of P% comes at the end of the list in the WRDCRAFT program.

Spacing the letters of a word in this program depends mostly, again, on having LF at the end of each character string, but we need even more spacing provisions in dealing with perpendiculars. For spacing between words, an additional LF (or another line-feed code for a space larger than the usual LF size) will be needed. You can repeat the line-feed code(s) several times for bigger spaces between words.

PROGRAM 7-2. WRDCRAFT. Spelling with perpendicular
homemade alphabets.

```

10 'PROGRAM 7-2: "WRDCRAFT" -- SPELLING PROGRAM FOR PERPENDICULARS
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR TRS-80 COMPUTERS AND PRINTERS
20 CLEAR 2000
25 DEFSTR A-Z                                '(ONLY VARIABLES WITH '%' TAG ARE NUMERIC)
30 P%=20: GOSUB 8000                          '(TO POSITIONING STRINGS)
90 SQ=CHR$(27)+CHR$(50)                      '1/72" SPACE
100 LF=CHR$(18)+CHR$(10)+CHR$(27)+CHR$(50)+CHR$(27)+CHR$(50) '9/72" LINE FEED
102 '-----
105 '***** UPPER-CASE ALPHABET (7 X 10 BOLD) AND PUNCTUATION *****
110 A=PS+STRING$(4,227)+CHR$(255)+CHR$(255)+CHR$(227)+CHR$(227)+CHR$(190)+CHR$(1
    56)+LF
120 B=PS+CHR$(191)+CHR$(255)+CHR$(227)+CHR$(227)+CHR$(191)+CHR$(191)+CHR$(227)+C
    HR$(227)+CHR$(255)+CHR$(191)+LF
130 C=PS+CHR$(190)+CHR$(255)+CHR$(227)+STRING$(4,131)+CHR$(227)+CHR$(255)+CHR$(1
    90)+LF

```

```

140 D=PS+CHR$(191)+CHR$(255)+STRING$(6,227)+CHR$(255)+CHR$(191)+LF
150 E=PS+STRING$(2,255)+STRING$(2,131)+STRING$(2,159)+STRING$(2,131)+STRING$(2,255)+LF
160 F=PS+CHR$(28)+CHR$(4)+CHR$(131)+CHR$(159)+CHR$(159)+CHR$(131)+CHR$(131)+CHR$(255)+CHR$(255)+LF
170 G=PS+CHR$(190)+CHR$(255)+CHR$(227)+CHR$(227)+CHR$(251)+CHR$(251)+CHR$(131)+CHR$(227)+CHR$(255)+CHR$(190)+LF
180 H=PS+STRING$(4,227)+STRING$(2,255)+STRING$(4,227)+LF
190 I=PS+STRING$(2,188)+STRING$(6,152)+STRING$(2,188)+LF
200 J=PS+CHR$(158)+CHR$(191)+STRING$(2,179)+STRING$(4,176)+STRING$(2,248)+LF
210 K=PS+CHR$(227)+CHR$(179)+CHR$(155)+CHR$(143)+CHR$(135)+CHR$(135)+CHR$(143)+CHR$(155)+CHR$(179)+CHR$(227)+LF
220 L=PS+STRING$(2,255)+STRING$(8,131)+LF
230 M=PS+STRING$(4,227)+CHR$(235)+CHR$(235)+CHR$(255)+CHR$(247)+STRING$(2,227)+LF
240 N=PS+CHR$(227)+CHR$(227)+CHR$(227)+CHR$(243)+CHR$(251)+CHR$(239)+CHR$(231)+CHR$(227)+CHR$(227)+CHR$(227)+LF
250 O=PS+CHR$(190)+CHR$(255)+STRING$(6,227)+CHR$(255)+CHR$(190)+LF
260 P=PS+STRING$(4,131)+CHR$(191)+CHR$(255)+STRING$(2,227)+CHR$(255)+CHR$(191)+LF
270 Q=PS+CHR$(144)+CHR$(190)+CHR$(191)+CHR$(251)+STRING$(4,227)+CHR$(255)+CHR$(190)+LF
280 R=PS+STRING$(4,227)+CHR$(191)+CHR$(191)+CHR$(227)+CHR$(227)+CHR$(255)+CHR$(191)+LF
290 S=PS+CHR$(190)+CHR$(255)+CHR$(227)+CHR$(224)+CHR$(254)+CHR$(191)+CHR$(131)+CHR$(227)+CHR$(255)+CHR$(190)+LF
300 T=PS+STRING$(8,152)+STRING$(2,254)+LF
310 U=PS+CHR$(190)+CHR$(255)+STRING$(8,227)+LF
320 V=PS+CHR$(136)+CHR$(156)+STRING$(3,182)+STRING$(5,227)+LF
330 W=PS+STRING$(2,227)+CHR$(247)+CHR$(255)+STRING$(2,235)+STRING$(4,227)+LF
340 X=PS+STRING$(2,227)+CHR$(247)+CHR$(190)+STRING$(2,156)+CHR$(190)+CHR$(247)+STRING$(2,227)+LF
350 Y=PS+STRING$(4,152)+CHR$(188)+CHR$(254)+STRING$(4,230)+LF
360 Z=PS+STRING$(2,255)+CHR$(131)+CHR$(134)+CHR$(140)+CHR$(152)+CHR$(176)+CHR$(24)+STRING$(2,255)+LF
370 PT=PS+STRING$(2,140)+CHR$(10) 'DECIMAL POINT, PERIOD
371 OB=PS+STRING$(2,158)+STRING$(6,134)+STRING$(2,158)+CHR$(10) 'OPEN BRACKET
372 CB=PS+STRING$(2,188)+STRING$(6,176)+STRING$(2,188)+CHR$(10) 'CLOSE BRACKET
373 SS=PS+CHR$(128)+CHR$(131)+CHR$(134)+CHR$(140)+CHR$(152)+CHR$(176)+CHR$(224)+CHR$(10) 'SLASH
380 HY=PS+STRING$(4,128)+STRING$(2,190)+LF$ 'HYPHEN, MINUS SIGN
390 '-----
400 '***** LOWER-CASE ALPHABET (7 X 10 BOLD) *****
410 AL=PS+STRING$(2,191)+CHR$(179)+CHR$(191)+CHR$(176)+CHR$(191)+CHR$(158)+CHR$(10)+SQ
420 BL=PS+CHR$(159)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(159)+STRING$(3,131)+CHR$(10)+SQ
430 CL=PS+CHR$(158)+CHR$(191)+CHR$(179)+CHR$(131)+CHR$(179)+CHR$(191)+CHR$(158)+CHR$(10)+SQ
440 DL=PS+CHR$(190)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(190)+STRING$(3,176)+CHR$(10)

```

```

450 EL=PS+CHR$(158)+CHR$(191)+CHR$(131)+CHR$(191)+CHR$(179)+CHR$(191)+CHR$(158)+
    CHR$(10)
460 FL=PS+STRING$(4,134)+STRING$(2,143)+CHR$(134)+CHR$(179)+CHR$(190)+CHR$(156)+
    CHR$(10)
470 GL=PS+CHR$(30)+CHR$(8)+CHR$(13)+CHR$(18)+CHR$(158)+CHR$(191)+CHR$(176)+CHR$(
    176)+CHR$(190)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(158)+CHR$(10)
480 HL=PS+STRING$(4,179)+CHR$(191)+CHR$(159)+STRING$(4,131)+CHR$(10)
490 IL=PS+STRING$(2,158)+STRING$(2,140)+CHR$(142)+CHR$(140)+CHR$(136)+CHR$(128)+
    STRING$(2,152)+CHR$(10)
500 JL=CHR$(142)+CHR$(159)+CHR$(155)+STRING$(4,152)+CHR$(156)+CHR$(152)+CHR$(144
    )+CHR$(128)+STRING$(2,152)+CHR$(10)
510 KL=PS+CHR$(179)+CHR$(155)+CHR$(143)+STRING$(2,135)+CHR$(143)+CHR$(155)+CHR$(
    179)+STRING$(2,131)+CHR$(10)
520 LL=PS+STRING$(2,158)+STRING$(6,140)+STRING$(2,142)+CHR$(10)
530 ML=PS+STRING$(5,235)+CHR$(255)+CHR$(182)+LF$
540 NL=PS+STRING$(5,179)+CHR$(191)+CHR$(159)+CHR$(10)
550 OL=PS+CHR$(158)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(158)+CHR$(10)
560 PL=STRING$(3,131)+CHR$(159)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(159)+CHR
    $(10)
570 QL=STRING$(3,176)+CHR$(190)+CHR$(191)+STRING$(3,179)+CHR$(191)+CHR$(190)+CHR
    $(10)
580 RL=PS+STRING$(4,131)+CHR$(151)+CHR$(191)+CHR$(155)+CHR$(10)
590 SL=PS+CHR$(158)+CHR$(191)+CHR$(176)+CHR$(156)+CHR$(131)+CHR$(191)+CHR$(158)+
    CHR$(10)
600 TL=PS+CHR$(156)+CHR$(190)+CHR$(182)+STRING$(2,134)+STRING$(2,159)+STRING$(3,
    134)+CHR$(10)
610 UL=PS+CHR$(190)+CHR$(191)+STRING$(5,179)+CHR$(10)
620 VL=PS+CHR$(136)+CHR$(156)+CHR$(190)+STRING$(4,182)+CHR$(10)+CHR$(27)+CHR$(50
    )
630 WL=PS+CHR$(182)+CHR$(255)+STRING$(5,235)+LF
640 XL=PS+STRING$(2,179)+CHR$(158)+CHR$(140)+CHR$(158)+STRING$(2,179)+CHR$(10)
650 YL=CHR$(134)+CHR$(143)+CHR$(152)+CHR$(156)+CHR$(190)+CHR$(183)+STRING$(4,179
    )+CHR$(10)
660 ZL=PS+STRING$(2,191)+CHR$(134)+CHR$(140)+CHR$(152)+STRING$(2,191)+CHR$(10)
690 ' ***** NUMBERS (7 X 10 BOLD) *****
695 ' -----
700 N0=PS+CHR$(190)+CHR$(255)+STRING$(6,227)+CHR$(255)+CHR$(190)+LF
710 N1=PS+STRING$(2,158)+STRING$(5,140)+CHR$(142)+CHR$(140)+CHR$(136)+LF
720 N2=PS+STRING$(2,255)+CHR$(131)+CHR$(134)+CHR$(188)+CHR$(248)+CHR$(224)+CHR$(
    227)+CHR$(255)+CHR$(190)+LF
730 N3=PS+CHR$(190)+CHR$(255)+CHR$(227)+CHR$(224)+STRING$(2,188)+CHR$(224)+CHR$(
    227)+CHR$(255)+CHR$(190)+LF
740 N4=PS+STRING$(3,176)+STRING$(2,255)+CHR$(179)+CHR$(182)+CHR$(188)+CHR$(184)+
    CHR$(176)+LF
750 N5=PS+CHR$(190)+CHR$(255)+CHR$(227)+STRING$(2,224)+CHR$(191)+CHR$(159)+CHR$(
    131)+STRING$(2,255)+LF
760 N6=PS+CHR$(190)+CHR$(255)+STRING$(2,227)+CHR$(255)+CHR$(191)+CHR$(131)+CHR$(
    227)+CHR$(255)+CHR$(190)+LF
770 N7=PS+STRING$(3,134)+CHR$(140)+CHR$(152)+CHR$(176)+STRING$(2,224)+STRING$(2,
    255)+LF
780 N8=PS+CHR$(190)+CHR$(255)+STRING$(2,227)+STRING$(2,190)+STRING$(2,227)+CHR$(
    255)+CHR$(190)+LF
790 N9=PS+CHR$(190)+CHR$(255)+CHR$(227)+CHR$(224)+CHR$(254)+CHR$(255)+STRING$(2,
    227)+CHR$(255)+CHR$(190)+LF

```

```

998 '-----
999 '***** TRANSLATIONS & PRINT COMMANDS *****
1000 READ ZZ: IF ZZ="END" THEN END
1001 IF VAL(ZZ)>9 THEN P%=VAL(ZZ): GOSUB 8000: GOTO 110
1002 IF ZZ="A" THEN LPRINT A;
1003 IF ZZ="B" THEN LPRINT B;
1004 IF ZZ="C" THEN LPRINT C;
1005 IF ZZ="D" THEN LPRINT D;
1006 IF ZZ="E" THEN LPRINT E;
1007 IF ZZ="F" THEN LPRINT F;
1008 IF ZZ="G" THEN LPRINT G;
1009 IF ZZ="H" THEN LPRINT H;
1010 IF ZZ="I" THEN LPRINT I;
1011 IF ZZ="J" THEN LPRINT J;
1012 IF ZZ="K" THEN LPRINT K;
1013 IF ZZ="L" THEN LPRINT L;
1014 IF ZZ="M" THEN LPRINT M;
1015 IF ZZ="N" THEN LPRINT N;
1016 IF ZZ="O" THEN LPRINT O;
1017 IF ZZ="P" THEN LPRINT P;
1018 IF ZZ="Q" THEN LPRINT Q;
1019 IF ZZ="R" THEN LPRINT R;
1020 IF ZZ="S" THEN LPRINT S;
1021 IF ZZ="T" THEN LPRINT T;
1022 IF ZZ="U" THEN LPRINT U;
1023 IF ZZ="V" THEN LPRINT V;
1024 IF ZZ="W" THEN LPRINT W;
1025 IF ZZ="X" THEN LPRINT X;
1026 IF ZZ="Y" THEN LPRINT Y;
1027 IF ZZ="Z" THEN LPRINT Z;
1028 IF ZZ="." THEN LPRINT PT;
1029 IF ZZ="(" THEN LPRINT OB;
1030 IF ZZ=")" THEN LPRINT CB;
1031 IF ZZ="/" THEN LPRINT SS;
1032 IF ZZ="-" THEN LPRINT HY;
1034 IF ZZ="*" OR ZZ="" THEN LPRINT LF;
1035 IF ZZ="a" THEN LPRINT AL;
1036 IF ZZ="b" THEN LPRINT BL;
1037 IF ZZ="c" THEN LPRINT CL;
1038 IF ZZ="d" THEN LPRINT DL;
1039 IF ZZ="e" THEN LPRINT EL;
1040 IF ZZ="f" THEN LPRINT FL;
1041 IF ZZ="g" THEN LPRINT GL;
1042 IF ZZ="h" THEN LPRINT HL;
1043 IF ZZ="i" THEN LPRINT IL;
1044 IF ZZ="j" THEN GOSUB 9000
1045 IF ZZ="k" THEN LPRINT KL;
1046 IF ZZ="l" THEN LPRINT LL;
1047 IF ZZ="m" THEN LPRINT ML;
1048 IF ZZ="n" THEN LPRINT NL;
1049 IF ZZ="o" THEN LPRINT OL;
1050 IF ZZ="p" THEN GOSUB 9000
1051 IF ZZ="q" THEN GOSUB 9000
1052 IF ZZ="r" THEN LPRINT RL;

```

```

1053 IF ZZ="s" THEN LPRINT SL;
1054 IF ZZ="t" THEN LPRINT TL;
1055 IF ZZ="u" THEN LPRINT UL;
1056 IF ZZ="v" THEN LPRINT VL;
1057 IF ZZ="w" THEN LPRINT WL;
1058 IF ZZ="x" THEN LPRINT XL;
1059 IF ZZ="y" THEN GOSUB 9000
1060 IF ZZ="z" THEN LPRINT ZL;
1061 REM -- USE 0 FOR 0 IN DATA
1062 IF VAL(ZZ)=1 THEN LPRINT N1;
1063 IF VAL(ZZ)=2 THEN LPRINT N2;
1064 IF VAL(ZZ)=3 THEN LPRINT N3;
1065 IF VAL(ZZ)=4 THEN LPRINT N4;
1066 IF VAL(ZZ)=5 THEN LPRINT N5;
1067 IF VAL(ZZ)=6 THEN LPRINT N6;
1068 IF VAL(ZZ)=7 THEN LPRINT N7;
1069 IF VAL(ZZ)=8 THEN LPRINT N8;
1070 IF VAL(ZZ)=9 THEN LPRINT N9;
1071 IF ZZ="Personal" THEN LPRINT P;EL;RL;SL;OL;NL;AL;LL;
1072 IF ZZ="Computer" THEN LPRINT C;OL;ML;PL;UL;TL;EL;RL;
1073 IF ZZ="Graphics" THEN LPRINT G;RL;AL;PL;HL;IL;CL;SL;
1074 GOTO 1000
1075 '***** DATA *****
1076 DATA Personal,*,Computer,*,Graphics,*
1077 DATA D,o,t,-,M,a,t,r,i,x,,P,r,i,n,t,e,r,,D,M,P,-,4,0,0,*
1078 DATA 20,B,0,0,K,25,L,0,0,K,30,K,0,0,K,25,T,0,0,K,20,N,0,0,K,*
1079 DATA A,B, ,C,D,E,,F,G,H,*,I,J, ,K,L,M,N,0,P,Q,R,S,T,U,V,W,X,Y,Z,*
1080 DATA a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,PT,OB,CB,SS,HY,*
1081 DATA 1,2,3,4,5,6,7,8,9,*
1082 DATA END
8000 '***** POSITIONING STRINGS *****
8010 QP%=FIX(P%/256): PP%=P%-256*QP%
8020 PS=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(QP%)+CHR$(PP%)
8030 RETURN
9000 '***** PRINTING OF LOWER-CASE DESCENDERS *****
9010 DP%=P%-13: QD%=FIX(DP%/256): DD%=DP%-256*QD%
9020 PD=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(QD%)+CHR$(DD%)
9030 IF ZZ="j" THEN LPRINT PD+JL;
9040 IF ZZ="p" THEN LPRINT PD+PL;
9050 IF ZZ="q" THEN LPRINT PD+QL;
9060 IF ZZ="y" THEN LPRINT PD+YL;
9070 RETURN

```

Programs of this sort could also use backspacing in the form of reverse line-feed codes (not to be confused with the CHR\$(8) code for horizontal backspacing). This would make it easy for the user to shorten the spacing between letters or numbers while entering data. In this perpendicular orientation, you see, the width of characters printed in a single printhead pass is fixed at seven dots for TRS-80 printers. That is more space than you need for skinny characters like letters I, i, or j, or number 1. You may want to make up for the too-

```

1999 '***** DATA *****
2000 DATA Personal,*,Computer,*,Graphics
2010 DATA D,O,t,-,M,a,t,r,i,x,,P,r,i,n,t,e,r,,D,M,P,-,4,0,0
2020 DATA 20,B,0,0,K,25,L,0,0,K,30,K,0,0,K,25,T,0,0,K,20,N,0,0,K
2030 DATA A,B, ,C,D,E,,F,G,H,*,I,J, ,K,L,M

```

Personal Computer Graphics Dot-Matrix Printer DMP-400

BOOKLOOKKOOKTOOKNOOK

AB CDE FGH IJ KLM

FIGURE 7-12. Spelling printouts for perpendiculars from the WRDCRAFT program.

large space that follows these as a result of the tacked-on LF that each character string has. So after these (or any) characters, you can type in a B1 (small), B2 (medium), or B3 (large) reverse line feed for more proportional spacing. This is the most flexible arrangement, offering a choice between monospaced and proportional characters at all times.

Figure 7-12 shows how WRDCRAFT prints a variety of DATA items.

Finally, if you have any need for printing reverse perpendicular words (that is, upside-down perpendiculars), such as in labeling the vertical axis of a scatterplot that runs parallel to the paper's edge, remember to spell the words backward.

Typography in BASIC

If you want characters larger than seven or eight dots wide in perpendicular printing or more than seven or eight dots high in upright or upside-down orientations, you'll have to print them in two or more passes of the printhead. Some impressive bold letters for the legend of a graph can be made with two passes, and even white-on-black prints of the letters will stay legible with photographic reductions greater than 80%.

With three passes you can produce letters that are big enough for major headings on newsletters, bulletin-board announcements, etc., and you have even more dots to play with in designing fancy lettering styles. Some three-pass examples are shown in figure 7-13.

A LINEDRAW type of program called TRISWEEP (program 7-3) was used for these letters. The binary-sum dot

CONDENSEDELITESTANDARDCONDENSED-ELONGATED

BEST BEST BEST BEST
BEST **BEST** **BEST** **BEST**

FIGURE 7-13. Three-pass homemade upright letters.

codes defining each of these four letters for the DMP-400, given in the DATA statements (lines 600–750), are stored in a different two-dimensional array for each letter. In lines 100–195 the codes for letter B are stored in an array called BC(R,C), in lines 200–305 those for E are stored in EC(R,C), and so forth. (More efficient programming for storing character codes in arrays is used in program 7-5.) The arrays are printed using nested FOR-NEXT loops (subroutine 1000).

When we go beyond three passes, we enter the enchanting realm of professional typography. Considering the drawings we saw in the last two chapters, it shouldn't be surprising that the dot-matrix printer can come up with some nice imitations of the typefaces used by the pros in the commercial and graphic arts. Better yet, methods of designing new lettering styles on a personal computer can be explored if we follow this line of thought.

PROGRAM 7-3. TRISWEEP. Forming large characters with three passes of the printhead.

```

10 'PROGRAM 7-3: "TRISWEEP" -- THREE-PASS HOMEMADE LETTERS
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 LF$=CHR$(18)+CHR$(13)+CHR$(27)+CHR$(51)           '7/72" ADJ. TO 22/216" LF
40 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(27);CHR$(31); 'CONDENSED, BOLD
50 DEFINT N,M,R,C,X
60 DIM BC(2,26),EC(2,26),SC(2,26),TC(2,26)           'FOR 2-DIMENSIONAL ARRAYS
70 INPUT "BLACK ON WHITE (B) OR WHITE ON BLACK (W)";BW$
100 '-----"B"-----
110 FOR R=0 TO 2
115   FOR C=0 TO 20
120     READ N: IF N=999 THEN GOTO 190
130     IF BW$="B" AND N>=0 THEN BC(R,C)=128+N: PRINT BC(R,C);: GOTO 180
140     IF N>=0 THEN BC(R,C)=255-N: PRINT BC(R,C);: GOTO 180
150     READ M
160     IF BW$="B" THEN FOR X=1 TO -N: BC(R,C)=128+M: C=C+1: NEXT X: GOTO 120
170     FOR X=1 TO -N: BC(R,C)=255-M: C=C+1: NEXT X: GOTO 120
180   NEXT C
190   PRINT "END OF ROW";R
195 NEXT R

```

```

200 '-----"E"-----
210 FOR R=0 TO 2
220   FOR C=0 TO 20
230     READ N: IF N=999 THEN GOTO 300
240     IF BW$="B" AND N>=0 THEN EC(R,C)=128+N: GOTO 290
250     IF N>=0 THEN EC(R,C)=255-N: GOTO 290
260     READ M
270     IF BW$="B" THEN FOR X=1 TO -N: EC(R,C)=128+M: C=C+1: NEXT X: GOTO 230
280     FOR X=1 TO -N: EC(R,C)=255-M: C=C+1: NEXT X: GOTO 230
290   NEXT C
300   PRINT "END OF ROW";R
305 NEXT R
310 '-----"S"-----
320 FOR R=0 TO 2
330   FOR C=0 TO 20
340     READ N: IF N=999 THEN GOTO 410
350     IF BW$="B" AND N>=0 THEN SC(R,C)=128+N: GOTO 400
360     IF N>=0 THEN SC(R,C)=255-N: GOTO 400
370     READ M
380     IF BW$="B" THEN FOR X=1 TO -N: SC(R,C)=128+M: C=C+1: NEXT X: GOTO 340
390     FOR X=1 TO -N: SC(R,C)=255-M: C=C+1: NEXT X: GOTO 340
400   NEXT C
410   PRINT "END OF ROW";R
415 NEXT R
420 '-----"T"-----
430 FOR R=0 TO 2
440   FOR C=0 TO 20
450     READ N: IF N=999 THEN GOTO 520
460     IF BW$="B" AND N>=0 THEN TC(R,C)=128+N: GOTO 510
470     IF N>=0 THEN TC(R,C)=255-N: GOTO 510
480     READ M
490     IF BW$="B" THEN FOR X=1 TO -N: TC(R,C)=128+M: C=C+1: NEXT X: GOTO 450
500     FOR X=1 TO -N: TC(R,C)=255-M: C=C+1: NEXT X: GOTO 450
510   NEXT C
520   PRINT "END OF ROW";R
525 NEXT R
530 '-----
540 GOSUB 1000
550 END
600 REM -- "B"
610 DATA 0,0,4,12,-2,124,12,-6,4,12,24,120,112,96,0,0,999
620 DATA -4,0,-2,127,-7,8,28,62,119,99,65,0,0,999
630 DATA 0,0,16,24,-2,31,24,-6,16,24,12,15,7,3,0,0,999
640 REM -- "E"
650 DATA 0,0,4,12,-2,124,12,-9,4,12,60,0,0,999
660 DATA -4,0,-2,127,-4,8,28,62,-8,0,999
670 DATA 0,0,16,24,31,31,24,-9,16,24,30,0,0,999
680 REM -- "S"
690 DATA 0,0,96,112,120,24,12,-7,4,8,24,48,120,0,0,999
700 DATA 0,0,65,3,7,12,-7,8,24,48,112,96,65,0,0,999
710 DATA 0,0,15,6,12,8,-7,16,24,12,15,7,3,0,0,999
720 REM -- "T"
730 DATA 0,0,60,12,-4,4,12,124,124,12,-4,4,12,60,0,0,999
740 DATA -9,0,-2,127,-9,0,999

```

```

750 DATA -7,0,16,24,31,31,24,16,-7,0,999
1000 '***** PRINTING OF WORD *****
1010 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(0);      'CARRIAGE RETURN
1020 FOR R=0 TO 2
1030   FOR C=0 TO 19: LPRINT CHR$(18);CHR$(BC(R,C));: NEXT C
1040   FOR C=0 TO 19: LPRINT CHR$(EC(R,C));: NEXT C
1050   FOR C=0 TO 19: LPRINT CHR$(SC(R,C));: NEXT C
1060   FOR C=0 TO 19: LPRINT CHR$(TC(R,C));: NEXT C
1070   LPRINT LF$;
1080 NEXT R
1090 RETURN

```

Whole-character programming. We're going to run into a good many curvy characters in the more decorative typefaces, so we'd better see how the TRS-80 and DMP-400 meet that kind of challenge first. Let's try the chic style that is well known to graphic artists as Brush (see figure 7-14).

Here again a modified LINEDRAW program is used. Each letter is printed in ten lines of printed dot patterns, and for each print line there is a line of DATA, ending, as usual, with a line-feed code, 999. The dot-pattern codes were figured out the same way as the WHITECAT codes, by tracing enlargements of the letters from a graphics catalogue and drawing a further enlargement of each letter on ten-squares-per-inch graph paper by hand. For the best resolution, each original square portion was distorted to a seven-by-ten rectangle in the latter enlargement to allow for condensed printing.

An even higher-resolution way of doing these letters is to use the FINEDRAW type of program introduced in chapter 6. For the Brush capital B, this would mean about 160 print-head passes instead of ten, with $\frac{1}{216}$ -inch line feeds between the passes (see the coding sheet in figure 7-15). Coding this may seem like six months at hard labor for just one silly letter, but remember that you have only two dot codes (binaries 0 and 8, for no dot and one dot) in each pass. The DATA statements for the Brush B are in program 7-4.

FIGURE 7-14. Letters of the Brush typeface, produced by a LINEDRAW type of program.



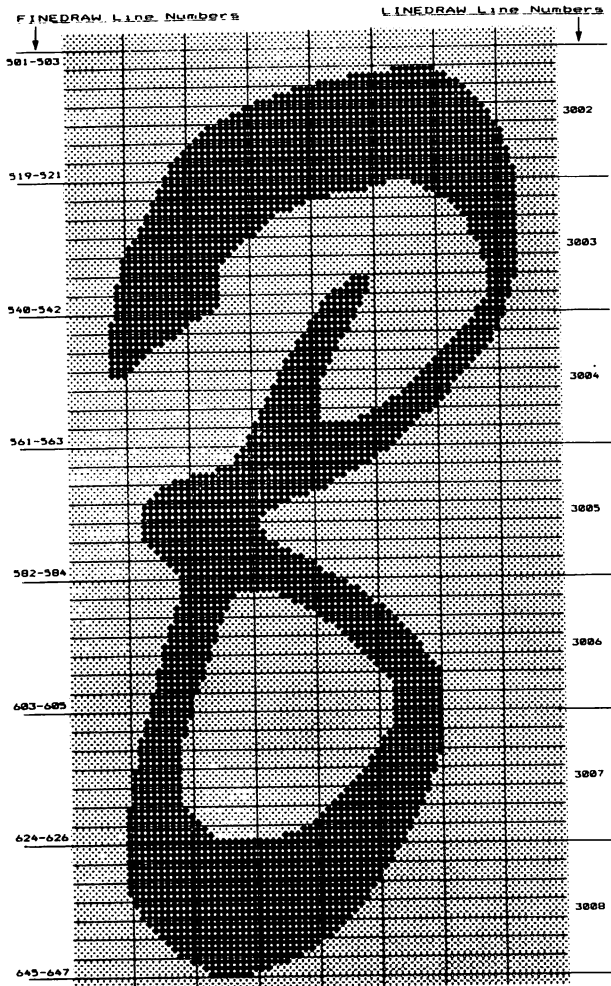


FIGURE 7-15. Coding sheet for the FINEDRAW version of the Brush B.

PROGRAM 7-4. BRUSHB. DATA codes for the Brush letter B.

```

10 'PROGRAM 7-4: "BRUSHB" -- DATA CODES FOR BRUSH LETTER B
20 'FOR TRS-80 OR IBM-EPSON PRINTERS
500 REM -- POSITION GIVEN BY FIRST VALUE IN EACH DATA LINE
501 DATA 100,-80,0,999,100,-80,0,999,100,-80,0,999,100,-80,0,999
502 DATA 100,-80,0,999
503 DATA 100,-80,0,999
504 DATA 100,-53,0,-7,8,-20,0,999
505 DATA 100,-46,0,-16,8,-18,0,999
506 DATA 100,-41,0,-23,8,-16,0,999
507 DATA 100,-37,0,-28,8,-15,0,999
508 DATA 100,-34,0,-32,8,-14,0,999

```

```

509 DATA 100,-31,0,-36,8,-13,0,999
510 DATA 100,-29,0,-39,8,-12,0,999
511 DATA 100,-27,0,-42,8,-11,0,999
512 DATA 100,-25,0,-44,8,-11,0,999
513 DATA 100,-23,0,-47,8,-10,0,999
514 DATA 100,-22,0,-49,8,-9,0,999
515 DATA 100,-21,0,-50,8,-9,0,999
516 DATA 100,-20,0,-51,8,-9,0,999
517 DATA 100,-19,0,-53,8,-8,0,999
518 DATA 100,-18,0,-54,8,-8,0,999
519 DATA 100,-17,0,-55,8,-8,0,999
520 DATA 100,-17,0,-56,8,-7,0,999
521 DATA 100,-16,0,-57,8,-7,0,999
522 DATA 100,-16,0,-36,8,-4,0,-17,8,-7,0,999
523 DATA 100,-15,0,-34,8,-9,0,-15,8,-7,0,999
524 DATA 100,-15,0,-28,8,-17,0,-13,8,-7,0,999
525 DATA 100,-14,0,-25,8,-23,0,-11,8,-7,0,999
526 DATA 100,-14,0,-23,8,-26,0,-10,8,-7,0,999
527 DATA 100,-13,0,-21,8,-30,0,-9,8,-7,0,999
528 DATA 100,-13,0,-20,8,-32,0,-8,8,-7,0,999
529 DATA 100,-12,0,-20,8,-33,0,-8,8,-7,0,999
530 DATA 100,-12,0,-19,8,-34,0,-8,8,-7,0,999
531 DATA 100,-11,0,-18,8,-37,0,-7,8,-7,0,999
532 DATA 100,-10,0,-18,8,-39,0,-6,8,-7,0,999
533 DATA 100,-10,0,-17,8,-40,0,-6,8,-7,0,999
534 DATA 100,-9,0,-17,8,-41,0,-6,8,-7,0,999
535 DATA 100,-9,0,-16,8,-43,0,-5,8,-7,0,999
536 DATA 100,-9,0,-16,8,-43,0,-5,8,-7,0,999
537 DATA 100,-9,0,-16,8,-21,0,-3,8,-19,0,-5,8,-7,0,999
538 DATA 100,-8,0,-17,8,-20,0,-4,8,-19,0,-4,8,-8,0,999
539 DATA 100,-8,0,-17,8,-18,0,-6,8,-19,0,-4,8,-8,0,999
540 DATA 100,-8,0,-17,8,-17,0,-6,8,-20,0,-4,8,-8,0,999
541 DATA 100,-8,0,-17,8,-16,0,-7,8,-19,0,-4,8,-9,0,999
542 DATA 100,-8,0,-16,8,-17,0,-6,8,-19,0,-5,8,-9,0,999
543 DATA 100,-8,0,-14,8,-18,0,-7,8,-19,0,-5,8,-9,0,999
544 DATA 100,-7,0,-13,8,-19,0,-7,8,-19,0,-5,8,-10,0,999
545 DATA 100,-7,0,-12,8,-20,0,-7,8,-18,0,-6,8,-10,0,999
546 DATA 100,-7,0,-11,8,-20,0,-7,8,-18,0,-6,8,-11,0,999
547 DATA 100,-7,0,-9,8,-21,0,-7,8,-18,0,-7,8,-11,0,999
548 DATA 100,-7,0,-7,8,-23,0,-7,8,-17,0,-7,8,-12,0,999
549 DATA 100,-7,0,-6,8,-23,0,-7,8,-17,0,-7,8,-13,0,999
550 DATA 100,-7,0,-5,8,-23,0,-8,8,-16,0,-7,8,-14,0,999
551 DATA 100,-7,0,-4,8,-24,0,-7,8,-16,0,-7,8,-15,0,999
552 DATA 100,-7,0,-3,8,-24,0,-8,8,-15,0,-7,8,-16,0,999
553 DATA 100,-34,0,-7,8,-15,0,-7,8,-17,0,999
554 DATA 100,-33,0,-8,8,-14,0,-7,8,-18,0,999
555 DATA 100,-33,0,-8,8,-13,0,-8,8,-18,0,999
556 DATA 100,-32,0,-9,8,-11,0,-9,8,-19,0,999
557 DATA 100,-32,0,-9,8,-10,0,-9,8,-20,0,999
558 DATA 100,-31,0,-10,8,-8,0,-10,8,-21,0,999
559 DATA 100,-31,0,-10,8,-6,0,-10,8,-23,0,999
560 DATA 100,-30,0,-26,8,-24,0,999
561 DATA 100,-30,0,-25,8,-25,0,999

```

562 DATA 100,-29,0,-25,8,-26,0,999
563 DATA 100,-29,0,-24,8,-27,0,999
564 DATA 100,-28,0,-24,8,-28,0,999
565 DATA 100,-27,0,-23,8,-30,0,999
566 DATA 100,-27,0,-22,8,-31,0,999
567 DATA 100,-24,0,-23,8,-33,0,999
568 DATA 100,-20,0,-25,8,-35,0,999
569 DATA 100,-17,0,-26,8,-37,0,999
570 DATA 100,-15,0,-27,8,-38,0,999
571 DATA 100,-14,0,-25,8,-41,0,999
572 DATA 100,-13,0,-23,8,-44,0,999
573 DATA 100,-12,0,-22,8,-46,0,999
574 DATA 100,-12,0,-20,8,-48,0,999
575 DATA 100,-12,0,-19,8,-49,0,999
576 DATA 100,-12,0,-19,8,-49,0,999
577 DATA 100,-12,0,-20,8,-48,0,999
578 DATA 100,-12,0,-20,8,-48,0,999
579 DATA 100,-13,0,-21,8,-46,0,999
580 DATA 100,-14,0,-22,8,-44,0,999
581 DATA 100,-15,0,-23,8,-42,0,999
582 DATA 100,-16,0,-24,8,-40,0,999
583 DATA 100,-17,0,-24,8,-39,0,999
584 DATA 100,-18,0,-25,8,-37,0,999
585 DATA 100,-18,0,-27,8,-35,0,999
586 DATA 100,-18,0,-29,8,-33,0,999
587 DATA 100,-18,0,-9,8,-7,0,-14,8,-32,0,999
588 DATA 100,-18,0,-8,8,-10,0,-13,8,-31,0,999
589 DATA 100,-17,0,-9,8,-12,0,-13,8,-29,0,999
590 DATA 100,-17,0,-8,8,-15,0,-12,8,-28,0,999
591 DATA 100,-17,0,-8,8,-16,0,-12,8,-27,0,999
592 DATA 100,-16,0,-8,8,-19,0,-11,8,-26,0,999
593 DATA 100,-16,0,-8,8,-20,0,-11,8,-25,0,999
594 DATA 100,-16,0,-8,8,-21,0,-11,8,-24,0,999
595 DATA 100,-15,0,-8,8,-23,0,-11,8,-23,0,999
596 DATA 100,-15,0,-8,8,-24,0,-10,8,-23,0,999
597 DATA 100,-15,0,-8,8,-25,0,-10,8,-22,0,999
598 DATA 100,-15,0,-7,8,-27,0,-10,8,-21,0,999
599 DATA 100,-14,0,-8,8,-28,0,-10,8,-20,0,999
600 DATA 100,-14,0,-8,8,-29,0,-9,8,-20,0,999
601 DATA 100,-14,0,-7,8,-31,0,-8,8,-20,0,999
602 DATA 100,-14,0,-7,8,-31,0,-8,8,-20,0,999
603 DATA 100,-13,0,-7,8,-32,0,-8,8,-20,0,999
604 DATA 100,-13,0,-7,8,-32,0,-8,8,-20,0,999
605 DATA 100,-13,0,-7,8,-32,0,-8,8,-20,0,999
606 DATA 100,-13,0,-7,8,-32,0,-8,8,-20,0,999
607 DATA 100,-12,0,-7,8,-33,0,-8,8,-20,0,999
608 DATA 100,-12,0,-7,8,-33,0,-8,8,-20,0,999
609 DATA 100,-12,0,-7,8,-32,0,-9,8,-20,0,999
610 DATA 100,-11,0,-7,8,-32,0,-10,8,-20,0,999
611 DATA 100,-11,0,-7,8,-32,0,-10,8,-20,0,999
612 DATA 100,-11,0,-7,8,-31,0,-11,8,-20,0,999
613 DATA 100,-11,0,-7,8,-30,0,-11,8,-21,0,999
614 DATA 100,-11,0,-7,8,-30,0,-11,8,-21,0,999

```

615 DATA 100,-10,0,-8,8,-29,0,-12,8,-21,0,999
616 DATA 100,-10,0,-8,8,-28,0,-12,8,-22,0,999
617 DATA 100,-10,0,-8,8,-27,0,-13,8,-22,0,999
618 DATA 100,-10,0,-8,8,-26,0,-14,8,-22,0,999
619 DATA 100,-10,0,-8,8,-25,0,-14,8,-23,0,999
620 DATA 100,-10,0,-8,8,-24,0,-15,8,-23,0,999
621 DATA 100,-9,0,-10,8,-22,0,-15,8,-24,0,999
622 DATA 100,-9,0,-11,8,-20,0,-16,8,-24,0,999
623 DATA 100,-9,0,-12,8,-18,0,-16,8,-25,0,999
624 DATA 100,-9,0,-13,8,-15,0,-18,8,-25,0,999
625 DATA 100,-9,0,-14,8,-11,0,-20,8,-26,0,999
626 DATA 100,-9,0,-45,8,-26,0,999
627 DATA 100,-9,0,-44,8,-27,0,999
628 DATA 100,-9,0,-43,8,-28,0,999
629 DATA 100,-9,0,-43,8,-28,0,999
630 DATA 100,-9,0,-42,8,-29,0,999
631 DATA 100,-9,0,-41,8,-30,0,999
632 DATA 100,-9,0,-41,8,-30,0,999
633 DATA 100,-9,0,-40,8,-31,0,999
634 DATA 100,-9,0,-39,8,-32,0,999
635 DATA 100,-10,0,-38,8,-32,0,999
636 DATA 100,-10,0,-37,8,-33,0,999
637 DATA 100,-11,0,-35,8,-34,0,999
638 DATA 100,-11,0,-34,8,-35,0,999
639 DATA 100,-12,0,-32,8,-36,0,999
640 DATA 100,-13,0,-30,8,-37,0,999
641 DATA 100,-14,0,-28,8,-38,0,999
642 DATA 100,-15,0,-25,8,-40,0,999
643 DATA 100,-16,0,-23,8,-41,0,999
644 DATA 100,-17,0,-20,8,-43,0,999
645 DATA 100,-19,0,-16,8,-45,0,999
646 DATA 100,-21,0,-11,8,-48,0,999
647 DATA 100,-22,0,-7,8,-51,0,999
648 DATA 100,-80,0,999
649 DATA 100,-80,0,999
650 DATA 100,-80,0,999
651 DATA 100,-80,0,999,100,-80,0,999,100,-80,0,999

```

Figure 7-16 shows what a FINEDRAW program can achieve in comparison to the LINEDRAW product and an actual dry-transfer copy from one of the catalogues. The lower examples are the results of shrinking the computer-produced letters on a Canon photocopier down to 65% (left) and 42% (right) of their printout size.

Clearly, at full size, the FINEDRAW product is better than the LINEDRAW version and is gaining fast on the dry-transfer rendition. And at reductions of more than 50% both the FINEDRAW and LINEDRAW lettering may pass as catalogue caliber.

Even if you don't agree, consider how handy a microcomputer and dot-matrix printer may be in designing new type-

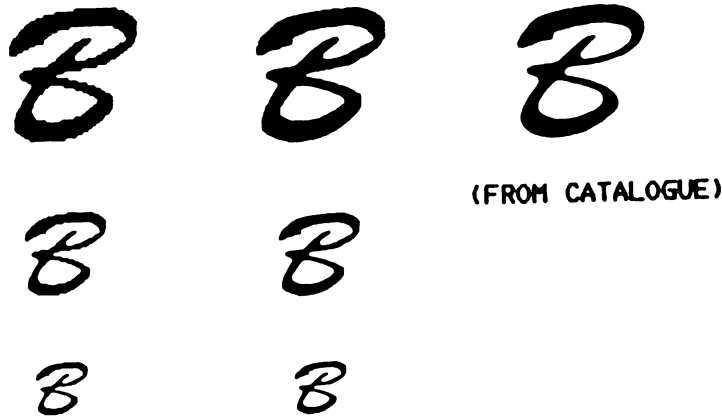
LINEDRAWFINEDRAWDRY-TRANSFER

FIGURE 7-16. Comparison of LINEDRAW (left) and FINEDRAW versions of the Brush B at 100%, 65%, and 42% of printout size.

faces. Characters could be roughed out using the lower-resolution LINEDRAW type of program, and any number of slight variations of a letter design could be saved as different programs on disk or tape under different filenames. Further refinement could be introduced in additional FINEDRAW versions, as printouts are examined from various distances and under a magnifying lens. Designing this way could beat all that erasing and redrawing by hand.

Figure 7-17 shows how convenient the editing process can be. An exploded view of the letter, produced by using twelve $\frac{1}{12}$ -inch line feeds between print lines, 5-CPI density, and white-on-black printing (the latter provides a good check on line lengths), can be laid out beside a listing of the DATA lines, for making changes easily.

Some further advantages come to the designer from the many different ways a letter can be printed, using different line feeds, densities, and reversals. See the examples in figure 7-18.

These examples of Brush characters illustrate what could be called whole-character programming, in which one character at a time is constructed and printed using READ and DATA statements. As with symbols and logos, this approach

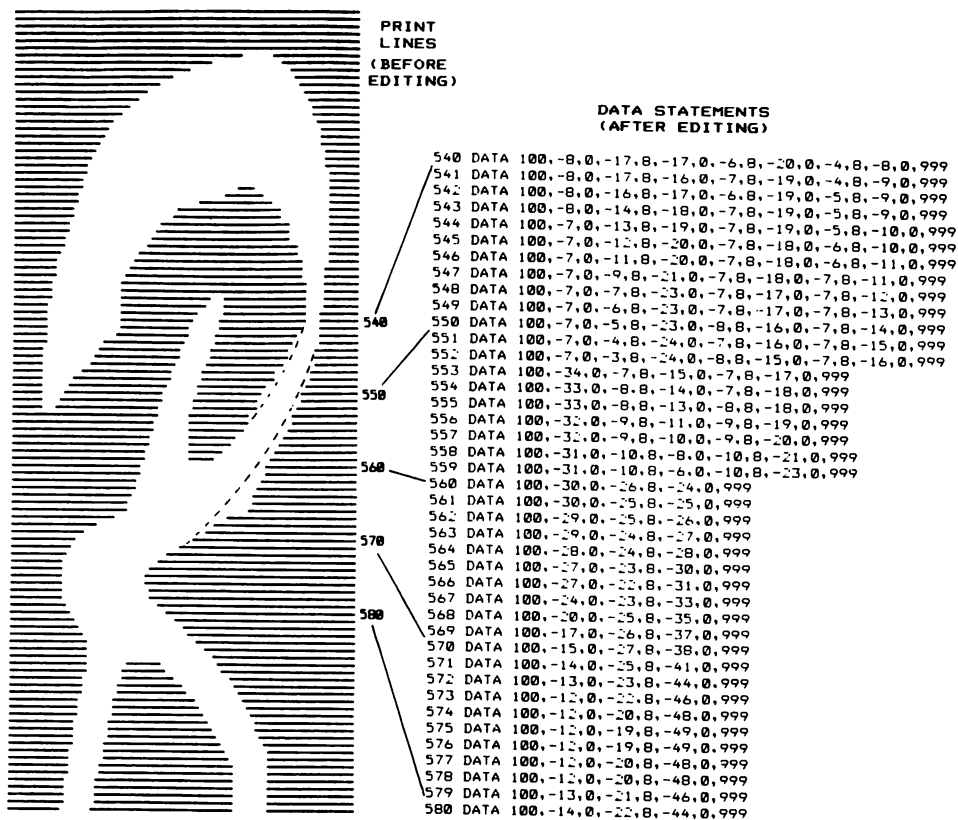


FIGURE 7-17. Exploded printout next to DATA lines for editing the Brush B.

may be fine for printing or designing single letters, but it is hardly the most convenient for dealing with whole alphabets, let alone putting letters together to form words or phrases.

Alphabetical arrays. If we can program the computer to print a single fancy letter in several printhead sweeps, then we ought to be able to store the dot codes somehow for all the characters we would use in spelling out words. In a whole-alphabet approach, we can read the dot codes in DATA statements into storage arrays—one array for each letter—instead of just reading the codes directly into LPRINT statements. Then we can form words in print by selecting the arrays and manipulate the printing of the words in many different ways.

Assuming that each letter will be printed in several print-head passes, the array for each letter needs to be a two-dimensional one. The array for capital A, to be printed in seven passes, would be aptly named AC(R,C), where AC stands for “letter A, capital,” R for rows (lines of print, 7), and C for

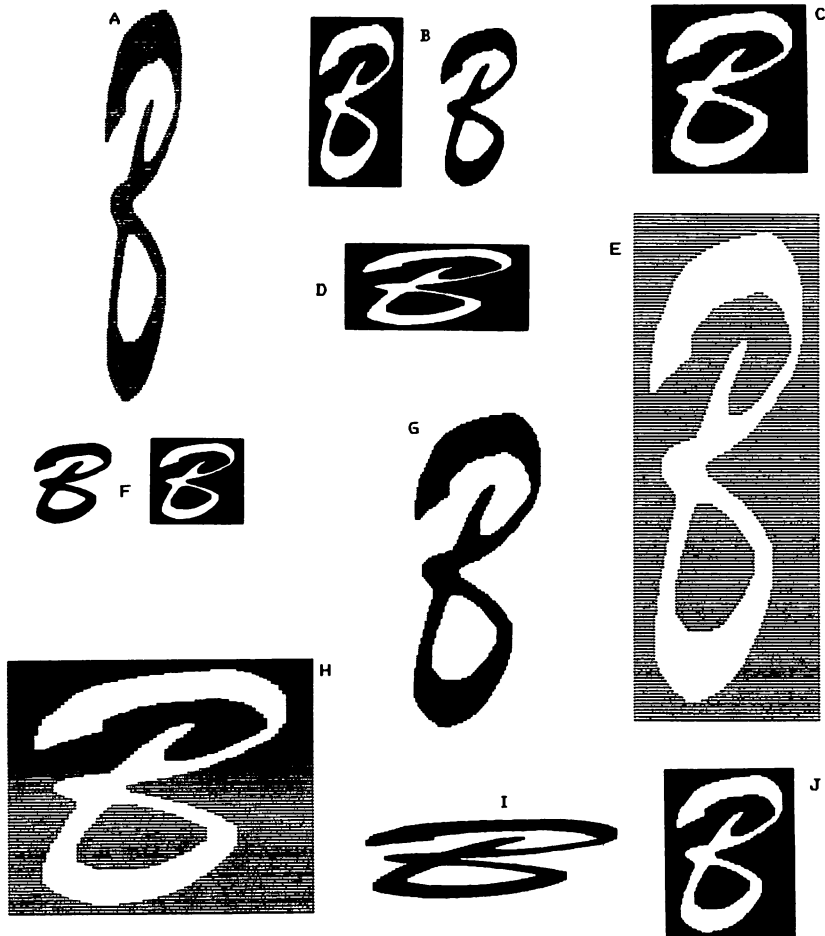


FIGURE 7-18. Varying densities and line spacing in printing the Brush B on the DMP-400. The same set of FINEDRAW codes was used in all printouts. Densities varied from condensed (A, C, F) to standard-elongated (H, I). Line-feed size varied from $\frac{1}{2}_{16}$ inch (D, F, I) to $\frac{5}{2}_{16}$ inch (E). (Notice the inconsistency of $\frac{4}{2}_{16}$ -inch line feeds in example H.)

columns (dot-column width of the letter A). For lowercase d it could be DL(R,C) and for the number 3 it could be N3(R,C). You could even have arrays for punctuation marks with names such as CP(R,C) for “colon, punctuation.”

You could have as many different characters of your typeface stored in arrays as your computer’s RAM space will allow, but watch out. If you have too many passes for each letter, both R (rows) and C (columns) may be so large, and thus each R-by-C array of dot codes may be so large, that

your memory space can't handle more than a part of the alphabet at a time. Even with only a few passes per letter, a 48K-RAM computer is likely to have room enough for capital letters and numbers but not lowercase letters as well. Disk storage of all the character codes and a program for reading the codes from a datafile (only part of the character set at a time) will be needed to cope with this problem.

In any event this space problem forces us to use binary-sum codes in the DATA lines. This is fine, because the simplest kind of program for storing the codes is a modified LINEDRAW program, which efficiently uses binary-sum codes anyway.

PROGRAM 7-5. TYPARRAY. Storage in arrays and spelling with homemade multipass letters (Helvetica Medium Italic typeface).

```

10 'PROGRAM 7-5: "TYPARRAY" -- ALPHABETICAL ARRAY PROGRAM, 'Aspen' EXAMPLE
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 DEFINT A-Z
40 DIM CH(7,60),AC(7,50),EL(7,43),NL(7,44),PL(7,45),SL(7,38)
50 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
60 LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18); 'CONDENSED, GRAPHIC MODE
70 GOTO 210
80 '***** SUBROUTINE FOR ARRAY STORAGE *****
90 FOR R=0 TO 7
100   FOR C=0 TO CL
110     READ N: IF N=999 THEN GOTO 180
120     IF BW$="B" AND N>=0 THEN CH(R,C)=128+N: GOTO 170
130     IF N>=0 THEN CH(R,C)=255-N: GOTO 170
140     READ M
150     IF BW$="B" THEN FOR X=1 TO -N: CH(R,C)=128+M: C=C+1: NEXT X: GOTO 110
160     FOR X=1 TO -N: CH(R,C)=255-M: C=C+1: NEXT X: GOTO 110
170   NEXT C
180 NEXT R
190 PRINT "END OF ";UL$;"-CASE LETTER ";LL$
200 RETURN
210 '***** STORAGE OF "A" *****
220 UL$="UPPER": LL$="A": CL=50 ' (50-COLUMN LETTER)
230 GOSUB 80
240 FOR R=0 TO 7: FOR C=0 TO CL: AC(R,C)=CH(R,C): NEXT: NEXT
250 '***** STORAGE OF "e" *****
260 UL$="LOWER": LL$="e": CL=43
270 GOSUB 80
280 FOR R=0 TO 7: FOR C=0 TO CL: EL(R,C)=CH(R,C): NEXT: NEXT
290 '***** STORAGE OF "n" *****
300 UL$="LOWER": LL$="n": CL=44
310 GOSUB 80

```

```

320 FOR R=0 TO 7: FOR C=0 TO CL: NL(R,C)=CH(R,C): NEXT: NEXT
330 '***** STORAGE OF "p" *****
340 UL$="LOWER": LL$="p": CL=45
350 GOSUB 80
360 FOR R=0 TO 7: FOR C=0 TO CL: PL(R,C)=CH(R,C): NEXT: NEXT
370 '***** STORAGE OF "s" *****
380 UL$="LOWER": LL$="s": CL=38
390 GOSUB 80
400 FOR R=0 TO 7: FOR C=0 TO CL: SL(R,C)=CH(R,C): NEXT: NEXT
410 GOSUB 50000 '(OR 10000, 20000, 30000, 40000 FOR OTHER WAYS OF PRINTING)
420 STOP
430 REM -- START OF DATA
440 REM ----- "A" -----
450 DATA -30,0,64,-12,96,-8,0,999
460 DATA -24,0,64,96,112,120,126,-15,127,64,-6,0,999
470 DATA -19,0,64,96,120,124,126,-6,127,63,31,15,3,1,0,63,-8,127,96,-5,0,999
480 DATA -14,0,96,112,120,126,-7,127,63,31,7,3,1,-7,0,31,-8,127,112,-4,0,999
490 DATA -8,0,64,96,112,120,126,-7,127,-19,63,-8,127,120,-3,0,999
500 DATA -2,0,64,96,112,120,124,-7,127,63,31,15,3,1,-20,0,7,-8,127,124,-2,0,999
510 DATA -51,0,999
520 DATA -51,0,999
530 REM ----- "e" -----
540 DATA -44,0,999
550 DATA -44,0,999
560 DATA -7,0,64,96,96,-2,112,-3,120,-5,124,126,-6,62,-3,126,-4,124,-2,120,-2,11
    2,96,64,-4,0,999
570 DATA -3,0,112,124,126,-7,127,111,103,99,97,97,-10,96,97,97,103,-10,127,120,-
    2,0,999
580 DATA -2,0,31,-9,127,115,67,-15,3,-10,67,-3,3,-2,0,999
590 DATA -3,0,1,3,7,15,15,31,31,63,63,-4,127,-3,126,-7,124,-2,126,-3,63,31,31,15
    ,15,7,3,3,1,-5,0,999
600 DATA -15,0,-7,1,-22,0,999
610 DATA -44,0,999
620 REM ----- "n" -----
630 DATA -45,0,999
640 DATA -45,0,999
650 DATA -9,0,96,-11,124,-2,112,-3,120,-5,124,-8,126,-2,124,120,96,-2,0,999
660 DATA -7,0,112,126,-8,127,63,15,7,3,-3,1,-6,0,97,-9,127,63,7,-3,0,999
670 DATA -4,0,64,120,-9,127,15,1,-11,0,112,126,-8,127,31,3,-5,0,999
680 DATA -2,0,96,124,-8,127,63,7,-11,0,96,120,-8,127,63,7,1,-7,0,999
690 DATA -45,0,999
700 DATA -45,0,999
710 REM ----- "p" -----
720 DATA -45,0,999
730 DATA -45,0,999
740 DATA -9,0,64,-11,124,-3,112,-2,120,-4,124,-6,126,-3,124,120,112,96,-3,0,999
750 DATA -7,0,96,124,-9,127,31,7,3,1,1,-8,0,1,71,-10,127,-2,0,999
760 DATA -5,0,96,124,-9,127,3,-11,0,64,96,112,124,-8,127,31,15,1,-2,0,999
770 DATA -3,0,96,126,-8,127,31,31,63,63,-3,126,-3,124,-4,126,-3,127,63,63,31,31,
    15,15,7,3,1,-6,0,999
780 DATA -1,0,112,126,-8,127,31,3,-8,0,-5,1,-19,0,999
790 DATA -10,3,1,-34,0,999
800 REM ----- "s" -----

```

```

810 DATA -39,0,999
820 DATA -39,0,999
830 DATA -7,0,64,96,112,120,120,124,124,-13,126,-5,124,120,120,112,96,64,-2,0,99
9
840 DATA -7,0,15,31,63,-7,127,121,-4,112,-5,96,65,67,-8,3,-2,0,999
850 DATA -2,0,-9,64,0,1,1,-4,3,-3,7,15,15,31,-7,127,126,126,124,-5,0,999
860 DATA -2,0,3,15,31,63,-6,127,126,124,124,-8,112,124,-4,127,63,63,31,15,7,3,-5
,0,999
870 DATA -10,0,-15,1,-14,0,999
880 DATA -39,0,999
10000 '***** PRINT WHOLE LETTER "A" *****
10010 LPRINT CHR$(18);          'ENTER GRAPHIC MODE
10020 FOR R=0 TO 7
10030   FOR C=0 TO 50
10040     LPRINT CHR$(AC(R,C));    'PRINT DOT PATTERN FROM ARRAY
10050   NEXT C: LPRINT CHR$(13);CHR$(27);CHR$(51);    'LINE FEED
10060 NEXT R
10070 RETURN
20000 '***** PRINT LETTERS A AND "s" BACKWARDS *****
20010 LPRINT CHR$(18);
20020 FOR R=0 TO 7
20030   FOR C=50 TO 0 STEP -1: LPRINT CHR$(AC(R,C));: NEXT C
20040   FOR C=38 TO 0 STEP -1: LPRINT CHR$(SL(R,C));: NEXT C
20050   LPRINT CHR$(13);CHR$(27);CHR$(51);    'LINE FEED
20060 NEXT R
20070 RETURN
20080 RETURN
30000 '***** PRINT STORED CODES FOR LETTER "A" *****
30010 LPRINT CHR$(30);          'EXIT GRAPHIC MODE
30020 FOR R=0 TO 7
30030   FOR C=0 TO 50
30040     LPRINT AC(R,C);          'PRINT DOT CODE
30050   NEXT C
30060   LPRINT CHR$(13);          'LINE FEED
30070 NEXT R
30080 RETURN
40000 '***** PRINT WHOLE LETTER "s" *****
40010 LPRINT CHR$(18);          'ENTER GRAPHIC MODE
40020 FOR R=0 TO 7
40030   FOR C=0 TO 44: LPRINT CHR$(EL(R,C));: NEXT C
40040   LPRINT CHR$(13);
40050 NEXT R
40060 RETURN
50000 '***** PRINT WORD "Aspen" *****
50010 LPRINT CHR$(18);          'ENTER GRAPHIC MODE
50020 FOR R=0 TO 7
50030   FOR C=0 TO 50: LPRINT CHR$(AC(R,C));: NEXT C
50040   FOR C=0 TO 38: LPRINT CHR$(SL(R,C));: NEXT C
50050   FOR C=0 TO 44: LPRINT CHR$(PL(R,C));: NEXT C
50060   FOR C=0 TO 43: LPRINT CHR$(EL(R,C));: NEXT C
50070   FOR C=0 TO 44: LPRINT CHR$(NL(R,C));: NEXT C
50080 LPRINT CHR$(13);CHR$(27);CHR$(51);    'LINE FEED
50090 NEXT R
50100 RETURN

```

Here's how it would work. Instead of immediately printing each dot code as it is read from a DATA line (or from disk storage), the modified program reads each set of codes into an array. Where the LINEDRAW program would say `IF N>=0 AND N<999 THEN LPRINT CHR$(128+N)`, the modified program would replace this with `IF N>=0 AND N<999 THEN AC(R,C)=128+N`. Where repeat codes (negative numbers) appear in the DATA, the modified program would become `FOR X=1 TO -N: AC(R,C)=128+M: C=C+1: NEXT X` instead of `LPRINT STRING$(-N,128+M)`, C being the subscript variable for the columns of the array.

The TYPARRAY program (program 7-5) operates this way. Helvetica Medium Italic (you've seen it everywhere) is the typeface that is coded in this example of the program's use. Five sets of dot codes, listed in DATA lines 430–880, are stored by lines 80–400 in five arrays for a capital letter (A) and four lowercase letters (e, n, p, and s). Notice how the widths of these letters vary, as indicated by the different values of CL in lines 220, 260, 300, 340, and 380.

Once you have the codes for several characters stored in (as many) arrays in this fashion, you can cash in on the BASIC language's powerful ways of manipulating arrays for the printing. With each letter represented by dot codes in a matrix of rows and columns, you have a choice: (1) printing one whole letter at a time; (2) printing one row at a time of each of the letters in a sequence, until all the letters in the sequence are completed; (3) printing one column at a time over a selected sequence of arrays; (4) printing a row, column, or whole letter backward or in whatever order you specify; and (5) printing any portion of a character you wish, instead of the whole character.

In ordinary printing you would be using (1) or (2), but typography designers might put the other three options to good use as well. Option (1), combined with some fancy line feeds and positioning commands, could provide interesting printouts with the letters of a word arranged on a page in the form of an arch, running along a diagonal, or some other arty way. For regular printing of horizontally aligned letters, (2) does the job. Time for some examples.

In TYPARRAY, FOR-NEXT loops are used in five subroutines to control the printing. Subroutine 10000 prints letter A, which has eight rows and fifty-one columns in its array, by



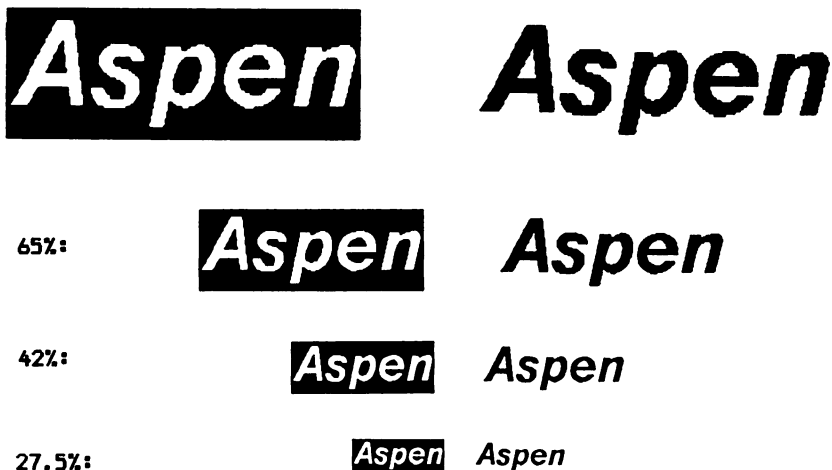
FIGURE 7-21. Backward A's and s's from the TYPARRAY program.

a time. By simply reversing the direction in the FOR statements for columns in these subroutines (to `FOR C=50 TO 0 STEP -1` in line 20030 and to `FOR C=38 TO 0 STEP -1` in line 20040), subroutine 20000 provides an example of option (3), backward printing, as shown in figure 7-21.

For option (2)—regular horizontal word formation—subroutine 50000 nests several FOR-NEXT loops (one for each letter) within the FOR-NEXT for rows. This produces letters like those in figure 7-22. That figure also shows how our rather rough-hewn Helveticas smooth out with photographic reduction.

If you have a spare week and want to do better than this, try an ultra-high-resolution (FINEDRAW) version of TYP-ARRAY, with $\frac{3}{216}$ -inch line feeds, 0 and 8 binary dot codes, and many more print and DATA lines. The amount of RAM space involved in printing the much larger arrays of characters in words would be pretty hefty, and storage of the dot codes on disk would almost certainly be necessary.

FIGURE 7-22. Smoothing out the rough edges by photographic reduction.



Letter-segment selection. If you take a close look at the professional typefaces in a typical graphics catalogue, you'll see that almost any of those character sets keeps its distinctive style and integrity by carefully repeating certain shapes or visual accents in many different letters. The bottom parts of the capital letters C, O, S, and U will have exactly the same curvature, for example, or the tops of B, D, P, and R will be exact duplicates in their form. And if you had to print several of these letters in the same word, you would want these common letter parts to be perfectly lined up horizontally.

This suggests another programming approach in typography that might be called letter-segment selection. Instead of whole-letter codes stored in DATA statements or arrays, codes for *parts* of letters that many different characters have in common can be stored in strings, and later these parts can be assembled in different combinations for different whole letters by combining the strings. Even though there are bound to be more different parts of letters in a whole alphabet than letters themselves, no matter how you slice them up, there could be advantages to this approach when it comes to programming word spelling in large titles or headings.

The best example for trying this out is probably the little-known Informal Gothic typeface (designed by Visual Graphics Corporation, Tamarac, Florida). This modern-but-informal lettering style was used by the Straus Printing Company (Madison, Wisconsin) for the front-cover titles of the author's *Baseball Graphics* and *Baseball's Pennant Races* books. This typeface is not a typical one, because its most distinctive feature is using lowercase letters as if they were capitals, which is also the main source of its charm.

This feature provides more than the usual number of characters having common letter parts. Even the top portions of the Informal Gothic A (shaped more like a) and E (e) have the same curvature as the tops of characters C, G, O, Q, S, 2, 3, 6, 8, and 9. Another feature, having many vertical edges in the characters that are parallel, provides more common segments in the interior portions of the letters. The age-old principle of computer programming—store anything that is often repeated in a subroutine or a string—seems to apply here.

In our BASIC program called STRAUS, we have planned for twelve passes of the printhead for each word or phrase in

which (up to about eighteen) characters are to be horizontally aligned. This produces black letters about 1.2 inches high, and each of these is constructed of twelve letter segments from the alphabet's total pool of seventy-five segments. The first (top) three segments of the letter D, for example, are produced by dot-pattern code sequences stored in strings called D1\$, D2\$, and D3\$ in a single subroutine that contains the code strings for all seventy-five segments.

After some "housekeeping" strings have also been defined (e.g., P1\$ for start-of-print position, LF\$ for line feed, BK\$ for a blank space), we can print a word by running a set of LPRINT statements such as those shown in program 7-6. (Again, the dollar signs for all of these string names can be omitted if DEFSTR A-Z is used.)

PROGRAM 7-6. STRAUS. String combinations for Informal Gothic letters.

```

10 'PROGRAM 7-6: "STRAUS" -- LETTER SEGMENTS FOR "PERSONAL" (TRS-80)
12 'SIMULATION OF INFORMAL GOTHIC TYPEFACE (VISUAL GRAPHICS CORP.)
14 '-----
303 REM -(POS.) P---E---R---S---O---N---A---L (L.FEED)
305 LPRINT P1$;D1$;G1$;D1$;G1$;G1$;N1$;G1$;L1$;LF$; 'TOP LINE
310 LPRINT P1$;D2$;G2$;D2$;G2$;G2$;N2$;G2$;L2$;LF$; 'LINE 2
315 LPRINT P1$;D3$;G3$;D3$;G3$;G3$;D3$;G3$;L2$;LF$; 'LINE 3
320 LPRINT P1$;O$ ;O$ ;O$ ;O$ ;O$ ;O$ ;O$ ;L2$;LF$; 'LINE 4
325 LPRINT P1$;O$ ;O$ ;B5$;S5$;O$ ;O$ ;A5$;L2$;LF$; 'LINE 5
330 LPRINT P1$;P6$;E6$;B6$;S6$;O$ ;O$ ;A6$;L2$;LF$; 'LINE 6
335 LPRINT P1$;P7$;H7$;B7$;S7$;O$ ;O$ ;A7$;L2$;LF$; 'LINE 7
340 LPRINT P1$;P8$;E8$;B8$;S8$;O$ ;O$ ;A8$;L2$;LF$; 'LINE 8
345 LPRINT P1$;C7$;O$ ;O$ ;O$ ;O$ ;O$ ;O$ ;L2$;LF$; 'LINE 9
350 LPRINT P1$;C7$;O$ ;O$ ;G0$;G0$;O$ ;G0$;L2$;LF$; 'LINE 10
355 LPRINT P1$;C7$;CE$;O$ ;CE$;CE$;O$ ;CE$;LE$;LF$; 'LINE 11
360 LPRINT P1$;C7$;CW$;RW$;CW$;CW$;O$ ;AW$;LW$;LF$; 'LINE 12
365 LPRINT P1$;BK$;CT$;BK$;CT$;CT$;BK$;AT$;LT$;LF$; 'LINE 13

```

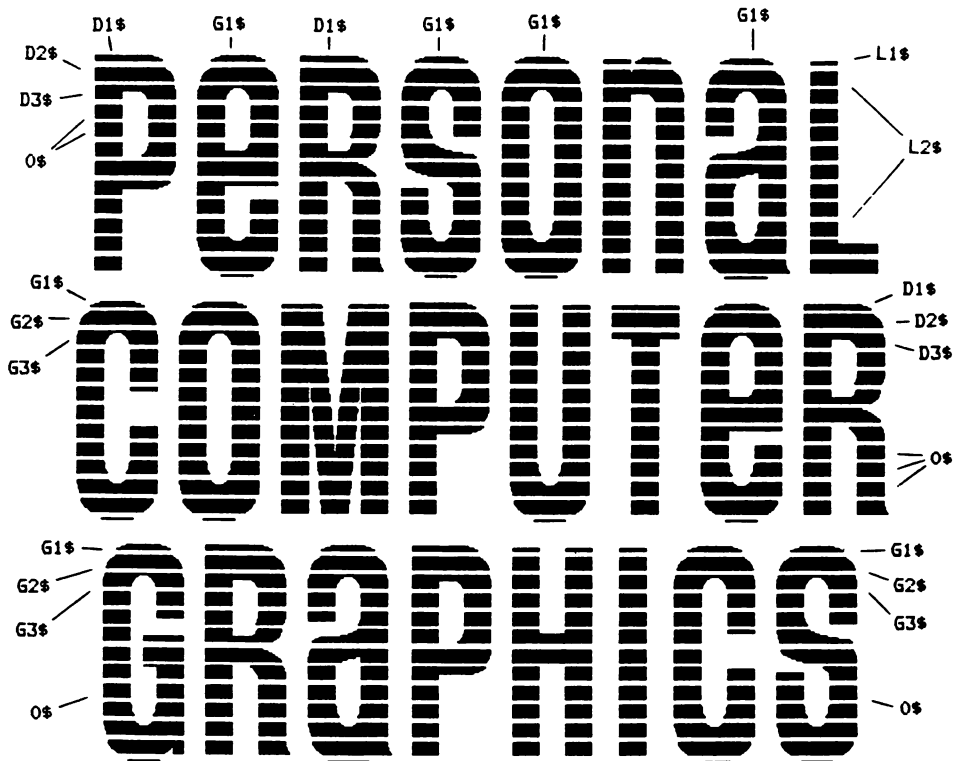
The printout and reversal are shown in figure 7-23. The white-on-black printout was done with another subroutine in which the letter-segments are defined by strings having black-white reversal codes for TRS-80 printers (see figure 4-3). Other renditions of the same letters, which a catalogue would call Informal Gothic Regular, Informal Gothic Condensed, and Informal Gothic Extended, would result from changes in print density.

For a better view of the letter segments themselves, let's explode some words by increasing the line and letter spacing. Figure 7-24 provides the exploded printout, and from



FIGURE 7-23. Informal Gothic letters printed in thirteen passes, using the letter-segment approach (DMP-400).

FIGURE 7-24. Exploding the letters shows some letter segments that are used repeatedly.



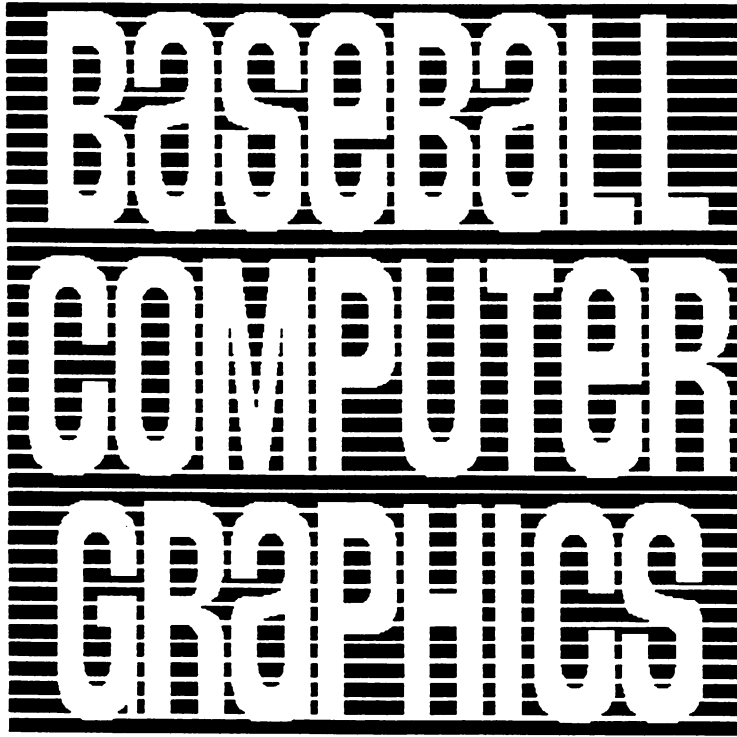


FIGURE 7-25. Black-white reversal of “Venetian blinds.”

this it’s easier to see that that shape called D1\$ in the STRAUS program (line 305, program 7-6) is used for the top of a P as well as for an R, a B, and the letter D itself. You can also see that the vertical sequence of G1\$, G2\$, and G3\$ gets used no fewer than eleven times in these three words.

By the way, if you stand back a bit and look at these words in a less analytical way, you’ll see that their explosion has created a whole new typeface having a certain amount of pizzazz by itself. This brilliant discovery deserves a new name—Venetian Blind, perhaps?

IBM users, of course, will probably wonder what is so original about this “discovery.” This exploded style is also interesting when reversed (see figure 7-25)—in that version the letters hang, unbroken, in front of the “Venetian blinds.” Maybe Big Blue’s logo would be refreshing that way, too.

Dot-style typefaces. There is a special group of typefaces which, you would think, were designed by and for computers—those having characters made out of dots. Examples are the

LEADERSHIP

LEADERSHIP in

ASTRA

ABCDEFGHIJKL

ABCDEFGHIJKL

ABCDEFGHIJKL
1234567890

PINBALL

POINTILLE

FIGURE 7-26. Five dot-style typefaces from the graphics catalogues.

Hollywood Lights and Sampler styles in the Formatt catalogue, the Astra and Pinball faces in Letraset's Letragraphica collection, and Chartpak's Pointille.

Figure 7-26 shows examples of these typefaces. On the dot-matrix printer most faces of this type are fun and easy to program, because each of them repeats a single element (a dot or star) that can be defined in a string as a simple combination of dot codes. Once defined in this way, a one-dimensional array is all you need for each character, with subscript values ranging from one to however many rows of dots or stars it takes to print the character. See how our HOLLYWRD (program 7-7) handles this.

PROGRAM 7-7. HOLLYWRD. Simulation of the dot-style Hollywood Lights typeface.

```

10 'PROGRAM 7-7: "HOLLYWRD" -- 'HOLLYWOOD LIGHTS' TYPEFACE
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND (WIDEBODY) PRINTERS
30 CLEAR 20000
40 PD=20: LPRINT CHR$(30);CHR$(27);CHR$(PD);      'PD=PRINT DENSITY
50 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
60 IF BW$="B" THEN GOTO 220
100 '***** GRAPHIC STRINGS FOR WHITE-ON-BLACK PRINTING *****
110 PS$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(140,255) 'BLACK FILL
120 PR$=STRING$(140,255)                                     ' " "
130 PT$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(20,255) ' " "
140 PU$=STRING$(20,255)                                       ' " "
150 PV$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(115,255) ' " "
160 PW$=CHR$(18)+STRING$(250,255)                             'BLACK SURROUND
170 ED$=CHR$(18)+STRING$(25,255)                              'BLACK SURROUND
180 DF$=CHR$(18)+CHR$(227)+CHR$(193)+CHR$(193)+STRING$(4,128)+CHR$(193)+CHR$(193)
    +CHR$(227)+CHR$(255)      'SQUARE CONTAINING OPEN CIRCLE

```

```

190 BK$=CHR$(18)+STRING$(11,255) 'SOLID BLACK SQUARE
200 LF$=CHR$(18)+CHR$(13) '7/72" LINE FEED
210 GOTO 300
220 '***** GRAPHIC STRINGS FOR BLACK-ON-WHITE PRINTING *****
230 PS$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(140,128) 'BLANK SPACE
240 PT$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(20,128) ' ' '
250 LF$=CHR$(18)+CHR$(13)+CHR$(27)+CHR$(50) ' ' '
260 PV$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+STRING$(115,128) ' ' '
270 DF$=CHR$(156)+CHR$(190)+CHR$(190)+STRING$(4,255)+CHR$(190)+CHR$(190)+CHR$(15
6)+CHR$(128) 'FILLED CIRCLE
280 BK$=CHR$(18)+STRING$(11,128) 'BLANK SQUARE
290 '***** SPELLING OF WORDS *****
300 DIM A$(11),B$(11),C$(11),D$(11),E$(11),F$(11),G$(11),H$(11),I$(11),J$(11),K$
(11),L$(11),M$(11),N$(11),O$(11),P$(11),Q$(11),R$(11),S$(11),T$(11),U$(11),V
$(11),W$(11),X$(11),Y$(11),Z$(11),EX$(11)
310 GOSUB 410 'MEMORIZE CHARACTER STRINGS
320 LPRINT PW$;PW$;PW$;ED$;LF$;
330 FOR I=0 TO 11: LPRINT PS$;H$(I);A$(I);P$(I);P$(I);Y$(I);PR$;LF$;: NEXT I
340 LPRINT PW$;PW$;PW$;ED$;
350 LPRINT CHR$(13);
360 LPRINT PW$;PW$;PW$;ED$;
370 FOR I=0 TO 11: LPRINT PT$;B$(I);I$(I);R$(I);T$(I);H$(I);D$(I);A$(I);Y$(I);PU
$;LF$;: NEXT I
380 LPRINT PW$;PW$;PW$;ED$;
390 LPRINT STRING$(3,13); 'TRIPLE LINE FEED-CARRIAGE RETURN
400 STOP
410 REM -- "A"
420 FOR I=2 TO 6: A$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
430 FOR I=9 TO 11: A$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
440 A$(0)=BK$+BK$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$
450 A$(1)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$
460 FOR I=7 TO 8: A$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
470 REM -- "B"
480 FOR I=0 TO 11 STEP 11: B$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$: NEXT I
490 FOR I=1 TO 10 STEP 9: B$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
500 FOR I=5 TO 6: B$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$: NEXT I
510 FOR I=2 TO 4: B$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
520 FOR I=7 TO 9: B$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
530 REM -- "C"
540 REM -- "D"
550 FOR I=2 TO 9: D$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
560 FOR I=1 TO 10 STEP 9: D$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
570 FOR I=0 TO 11 STEP 11: D$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$: NEXT I
580 REM -- "E"
590 REM -- "F"
600 REM -- "G"
610 REM -- "H"
620 FOR I=0 TO 4: H$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
630 FOR I=5 TO 6: H$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
640 FOR I=7 TO 11: H$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
650 REM -- "I"
660 FOR I=0 TO 11: I$(I)=BK$+DF$+DF$+DF$+BK$: NEXT I
670 REM -- "J"
680 REM -- "K"

```

```

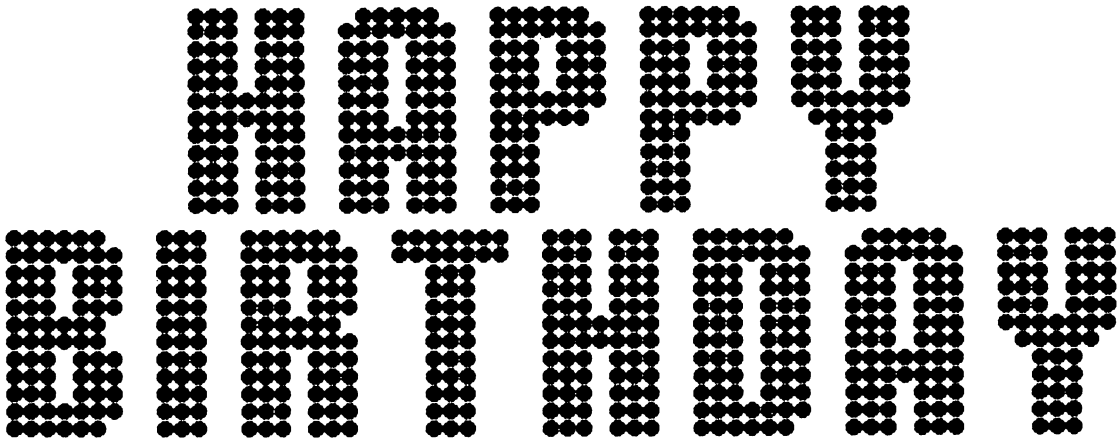
690 REM -- "L"
700 REM -- "M"
710 REM -- "N"
720 REM -- "O"
730 REM -- "P"
740 FOR I=7 TO 11: P$(I)=BK$+DF$+DF$+DF$+BK$+BK$+BK$+BK$: NEXT I
750 FOR I=2 TO 4: P$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
760 FOR I=0 TO 6 STEP 6: P$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$: NEXT I
770 FOR I=1 TO 5 STEP 4: P$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
780 REM -- "Q"
790 REM -- "R"
800 FOR I=0 TO 5 STEP 5: R$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+BK$+BK$: NEXT I
810 FOR I=1 TO 6 STEP 5: R$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
820 FOR I=2 TO 4: R$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
830 FOR I=7 TO 11: R$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
840 REM -- "S"
850 REM -- "T"
860 FOR I=0 TO 1: T$(I)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$: NEXT I
870 FOR I=2 TO 11: T$(I)=BK$+BK$+BK$+DF$+DF$+DF$+BK$+BK$+BK$: NEXT I
880 REM -- "U"
890 REM -- "V"
900 REM -- "W"
910 REM -- "X"
920 REM -- "Y"
930 FOR I=0 TO 6: Y$(I)=BK$+DF$+DF$+DF$+BK$+DF$+DF$+DF$+BK$: NEXT I
940 FOR I=7 TO 11: Y$(I)=BK$+BK$+BK$+DF$+DF$+DF$+BK$+BK$+BK$: NEXT I
950 Y$(5)=BK$+DF$+DF$+DF$+DF$+DF$+DF$+DF$+BK$
960 Y$(6)=BK$+BK$+DF$+DF$+DF$+DF$+DF$+BK$+BK$
970 REM -- "Z"
980 REM -- EXCLAMATION POINT -- EX$(I)
990 FOR I=0 TO 8: EX$(I)=BK$+DF$+DF$+DF$+BK$: NEXT I
1000 EX$(9)=BK$+BK$+BK$+BK$+BK$
1010 FOR I=10 TO 11: EX$(I)=BK$+DF$+DF$+DF$+BK$: NEXT I
1020 RETURN

```

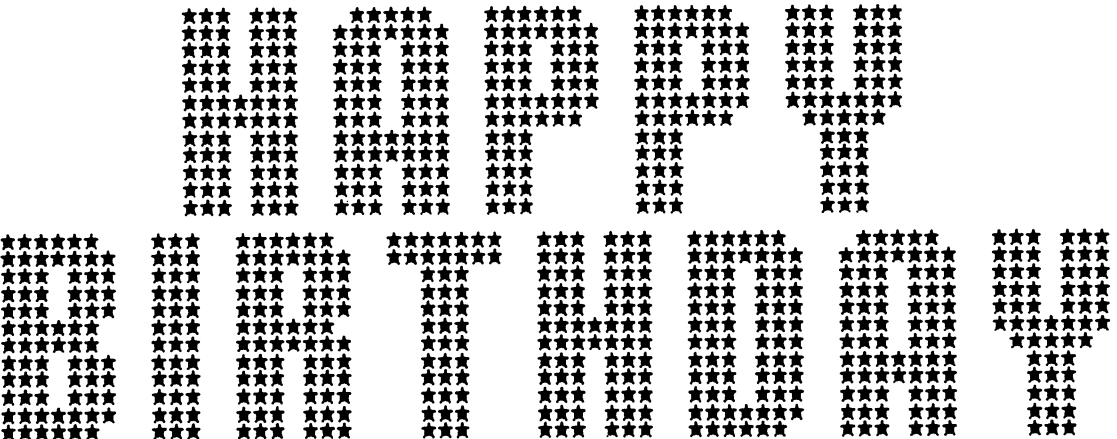
Figure 7-27 shows the simulation of the Hollywood Lights typeface that the HOLLYWRD program provides, and it also shows how a phony (and inaccurate) version of Astra can be produced by substituting codes for a star in place of the black-circle codes. The third HAPPY BIRTHDAY is a more authentic Astra example, produced by defining the character strings, printing the groups of stars in eleven passes instead of twelve, and spacing them farther apart. With only minor changes in the HOLLYWRD program, you can go from Astra to Sampler to Pointille with ease, and on to further creations.

And now, for those of you who have been viewing these last three chapters as a never-ending digression from the main business at hand, relief is only a page away. We finally come to the programming of *graphs*!

Hollywood Lights:



Hollywood Lights with stars:



Astra:

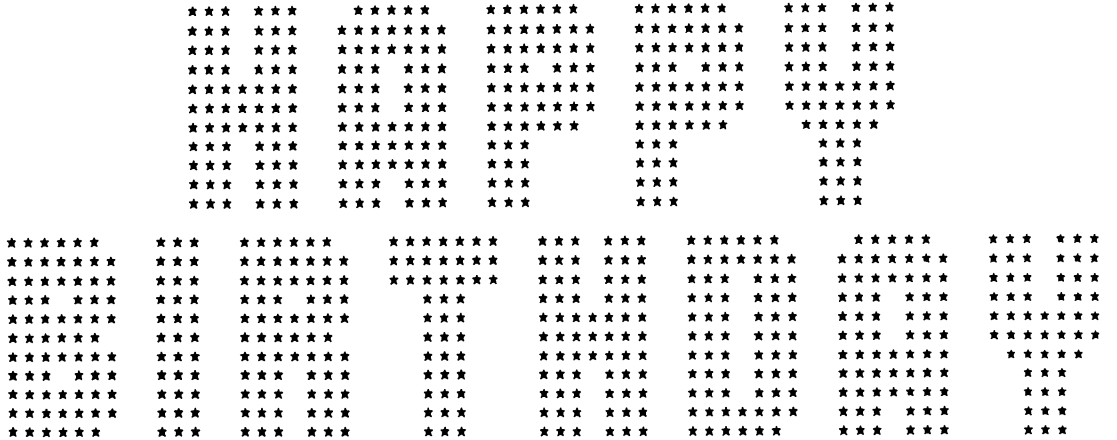


FIGURE 7-27. Three printouts from the HOLLYWRD program.

8

Bar Graphs

YOU'D think that a bar graph is a bar graph and that's all there is to it. But there are horizontal and vertical bar graphs, anchored and floating bar graphs, one-way and two-way bar graphs, segmented and unsegmented bar graphs, and lots of combinations. That means that we have our work cut out for us.

Horizontal Bar Graphs

The first thing we'll discover is that horizontal bar graphs are easier to program than vertical bar graphs. From the standpoint of printer graphics, the important difference between horizontal and vertical bar graphs is not whether the bars run parallel to or at right angles to the printhead's direction—in this chapter they'll run parallel for both types. Rather, it is a matter of whether the bars run parallel or perpendicular to most of the numbering and labeling in the graph. The more we can make use of the printer's built-in characters for the numbering and lettering, the easier the programming will be, and it is the horizontal bar graphs that have the bars running parallel to those (upright-oriented) characters.

A simple bar graph. Our first few examples of each type of bar graph will make this clearer. In figure 8-1, we have a simple horizontal bar graph produced by the BARGRAPH program for IBM-Epson printers (program 8-1). Each of the solid bars in the graph is printed by simply repeating the

REPORTED DEATHS FROM BOREDOM IN THE UNITED STATES, 1990-1995

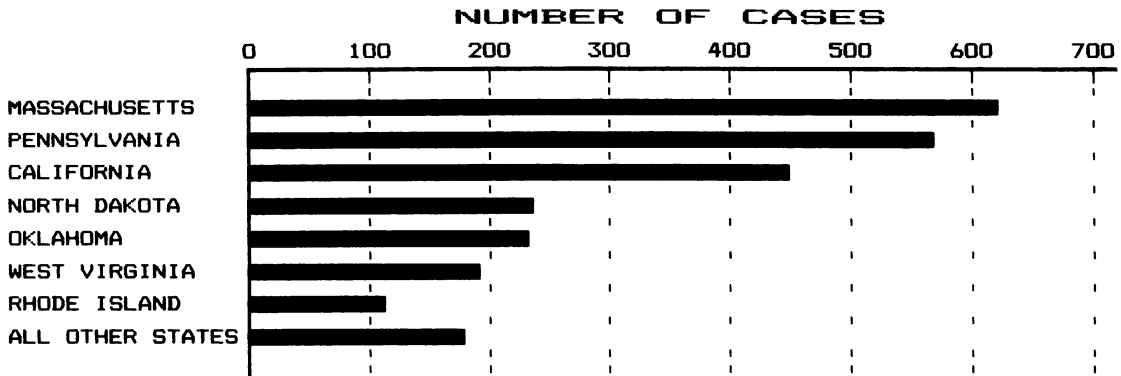


FIGURE 8-1 A simple horizontal bar graph from the BARGRAPH program for the DMP-400.

single 8-dot code CHR\$(255). The length of each bar indicates the data value, and to get those bar lengths as accurate as possible, the program uses the double-density graphic mode. Before each bar is printed, its data value is converted in line 1020 to the N1 and N2 values (here called V1% and V2%) that state precisely how many dot columns are reserved for the bar. Line 140 then prints the bar by following the 27+"L"+N1+N2 provision with STRING\$(N,255) commands. The N values in these STRING\$ functions must be equal to 255 or 0, and S1 or S2, as calculated by lines 1030-1050.

PROGRAM 8-1. BARGRAPH. Horizontal bar graph.

```

10 'PROGRAM 8-1: "BARGRAPH" -- SIMPLE BAR GRAPH FOR IBM-EPSON PRINTERS
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR IBM-EPSON PRINTERS AND TRS-80 COMPUTERS (MODEL III: USE BOOTHE'S DRIVER)
20 CLEAR 1000
30 LPRINT CHR$(27);"@"; 'MASTER RESET
40 CR$=CHR$(13) 'CARRIAGE RETURN
50 SF=1 'SF=SCALE FACTOR
60 LF$=CHR$(27)+"A"+CHR$(8)+CHR$(10) '8/72" LINE FEED
70 PX$=CHR$(27)+"L"+CHR$(199)+CHR$(0)+STRING$(199,0)
80 GOSUB 5000 'HORIZ. AXIS NUMBERING, LABELING
90 FOR N=1 TO 3: GOSUB 2000 : NEXT N 'START OF VERTICAL AXIS
100 READ N$,V
110 IF N$="END OF DATA" THEN FOR N=1 TO 2: GOSUB 2000 : NEXT N: GOTO 999
120 LPRINT CHR$(27);"E";N$;CR$;PX$;CHR$(27);"L";CHR$(2);CHR$(0);STRING$(2,255);
'PRINTING OF BAR LABEL AND SIDE-AXIS SEGMENT
130 GOSUB 1000 'NUMERICAL CONVERSIONS OF V
140 LPRINT CHR$(27);"L";CHR$(V1%);CHR$(V2%);STRING$(S1,255);STRING$(S2,255);STR
NG$(V1,255);STRING$(V2,255);
'PRINTING OF BAR
150 LPRINT CR$;LF$; 'CARRIAGE RETURN, LINE FEED

```

```

160 GOSUB 2000 'PRINTING BETWEEN BARS (SIDE AXIS, VERT. DASHES)
170 GOTO 100
500 'DATA FORMAT: N$=BAR LABEL (UP TO 16 CHARACTERS), V=NUMERICAL VALUE
510 DATA "REPORTED DEATHS FROM BOREDOM IN THE UNITED STATES, 1990-1995", "NUMBER
    OF CASES"
520 DATA MASSACHUSETTS,621,PENNSYLVANIA,568,CALIFORNIA,449,NORTH DAKOTA,236,OKLA
    HOMA,232,WEST VIRGINIA,191,RHODE ISLAND,112,ALL OTHER STATES,178,END OF DATA
    ,0
999 PRINT "JOB DONE": END
1000 '***** NUMERICAL CONVERSIONS *****
1010 V=SF*V 'ADJUST V BY SCALE FACTOR, SF
1020 V2%=FIX(V/256): V1%=FIX(V-256*V2%) 'FOR NO. OF COLUMNS OF GRAPHIC SPACE TO
    RESERVE FOR PRINTING OF A BAR
1030 IF V2%=0 THEN S1=0: S2=0
1040 IF V2%=1 THEN S1=255: S2=0
1050 IF V2%=2 THEN S1=255: S2=255
1060 RETURN
2000 '***** VERTICAL-AXIS SEGMENT WITH LINE FEED *****
2010 LPRINT PX$;CHR$(27);"L";CHR$(2);CHR$(0);STRING$(2,255);
2020 FOR N=1 TO 7: LPRINT CHR$(27);"L";CHR$(100);CHR$(0);STRING$(99,0);CHR$(60);
    : NEXT N
2030 LPRINT CR$;LF$;
2040 RETURN
5000 '***** HORIZONTAL-AXIS HASHMARKS, NUMBERING, LABELING *****
5010 READ T$: LPRINT CHR$(27);"E";TAB(9);T$;CR$;LF$;LF$; 'PRINTING OF TITLE
5020 READ YL$: LPRINT CR$;TAB(34);CHR$(27);"W1";YL$;CHR$(27);"W0";
5030 LPRINT CR$;LF$;LF$; 'CARRIAGE RETURN, DOUBLE LINE FEED
5040 LPRINT PX$;CHR$(8);CHR$(27);"L";CHR$(8);CHR$(0);STRING$(8,0);"0";CHR$(27);"
    L";CHR$(76);CHR$(0);STRING$(76,0);"100"; 'START OF TOP-AXIS NUMBERING
5050 SP$=CHR$(27)+"L"+CHR$(64)+CHR$(0)+STRING$(64,0): LPRINT SP$;"200";SP$;"300"
    ;SP$;"400";SP$;"500";SP$;"600";SP$;"700"; 'NUMBERS 200-700
5060 LPRINT CR$;LF$;
5070 LPRINT CHR$(27);"J";CHR$(11); 'EXTRA LINE FEED (11/216")
5080 LPRINT CR$;PX$;CHR$(27);"L";CHR$(209);CHR$(2);STRING$(250,192);STRING$(250,
    192);STRING$(221,192); 'TOP-AXIS LINE
5090 LPRINT CR$;PX$;CHR$(27);"L";CHR$(2);CHR$(0);STRING$(2,255); 'HASHMARK
    AT ZERO POINT
5100 FOR N=1 TO 7: LPRINT CHR$(27);"L";CHR$(100);CHR$(0);STRING$(99,0);CHR$(127)
    ;: NEXT N 'REMAINING HASHMARKS
5110 LPRINT CR$;LF$;
5120 RETURN

```

Since the BARGRAPH program allots space for up to sixteen characters for labeling each bar immediately to its left, the longest any bar can be is 720 dot columns (six inches on the printed page). You can reduce this allotment to allow room for printing each bar's numerical value at its right-hand end.

Printing those values, as with the bar labels on the left and the scale numbering and title at the top of the graph, can all be done with the printer's built-in characters. This is a good

deal easier than having to define and position homemade characters.

If you study subroutine 5000, however, you'll see that horizontal graphs are still not all that easy. Here we have our first of many examples showing that the programming of a graph's main axis (the top axis in a horizontal graph, the Y or vertical axis in a vertical graph) is almost as much work as all the other ingredients of a graph combined. After the graph's title (T\$) is read and printed (line 5010), the main axis label (YL\$) is printed in expanded (5-CPI) mode (line 5020). Then the numbering of the top axis is done (lines 5040–5050), using built-in numbers in pica pitch. All the three-digit numbers are separated by sixty-four dot columns, and since this is not an even multiple of six or twelve, the graphic mode is used for precise spacing. After the numbering and more line feeds, the axis itself is drawn (line 5080), using the top two dots of the printhead (dot code 192). Finally, hash marks lined up with the center of the central digit in each scale number are printed (lines 5090–5100), using a seven-dot code, CHR\$(127).

Scaling of the main axis. All this programming cannot be done, of course, until after the scale of the axis is calculated. In nearly every kind of graph, this has to be done, and it's a good idea to lay out a worksheet for the scaling so you can see exactly how many dot columns of space are available for the plotting of data values in the graph after allowing for labeling space, a title, perhaps a frame around the graph, and any other features outside the plotting area.

Don't make hard work out of scaling. There's no higher mathematics involved, and most of the time it's a simple matter requiring only a little common sense. Once the plotting space on the paper is determined, you need to find the range of data values to be plotted and decide how close you'll let the highest and lowest values come to the axes or margins of the graph for a good overall appearance. When you know both the number of dot columns available and the range of numerical values, you can calculate how many dot columns it takes to divide the axis into several equal segments that will suit the numerical scales of your data values. In our BARGRAPH illustration, the 700-plus dot columns available (after labeling space on the left) were divided into seven

equal segments of 100 dot columns and gave us a handy one-to-one relationship of data values to dot columns, since the highest data value (for Massachusetts) was between 600 and 700 on the “number of cases” scale. If the number of cases had extended to over 1,200, then a two-to-one ratio would have worked well, with the axis numbered from 0 to 1400 instead of from 0 to 700. The variable SF (scale factor) in lines 50 and 1010 amounts to the same thing as this ratio of data values to dot columns.

There’s a lot more to scaling than this, since data values can include negative values, be bunched together in ways that make it difficult to show differences clearly, and present other challenging problems. Some of our later examples will show how to cope with these things, or a standard book on designing graphs can be consulted. In commercial software for common types of bar and line graphs, you’ll find “auto-scaling”—automatic scaling and numbering of the main axis based directly on the range of data values. This involves scanning of all data values by the computer before anything is plotted, determining the highest and lowest data values in the whole set, and calculating the difference between them. Scale values appearing as numbers on the axis are then selected in subroutines in accordance with the maximum, minimum, and range of the data values and the ratio of the range to dot columns available.

Beyond scaling and numbering, there is the matter of printing an axis and its hash marks. An assortment of styles for axes that run parallel to the printhead’s movement (i.e., the top axis in a horizontal graph, or the Y axis in a vertical graph) are displayed in figure 8-2. The BASIC statements that accompany the drawings are designed for TRS-80 printers, but for other printers you need only replace `CHR$(18)` with a graphic reservation command (the `27 + “K” + N1 + N2` sequence for IBM-Epson printers or the `27 + “Gnnnn”` sequence for Apple printers), substitute positioning codes for the TRS-80’s `27 + 16 + N1 + N2` sequence, and convert the dot codes using figure 4-1.

For the axis lines themselves, using at least a two-dot line is recommended if you’re concerned about how well a graph will reproduce photographically. If you like heavier styles, three-, four-, or five-dot thicknesses can be used, but if you

Y-AXIS SUBROUTINES

```

10000 '***** Y AXIS WITH FOR-NEXT HASHMARKS *****
10005 REM -- 'CLEAR 1000' NEEDED IN PROGRAM
10010 LPRINT CHR$(18);STRING$(50,128);STRING$(201,131);STRING$(200,131);
401-DOT-LENGTH Y AXIS STARTING AT COLUMN 50
10020 FOR Y=50 TO 450 STEP 100: Y1%=Y/256: Y2%=Y-256*Y1%: LPRINT CHR$(27);CHR$(1
6);CHR$(Y1%);CHR$(Y2%);CHR$(252);: NEXT Y 'LARGE HASHMARKS
10030 FOR Y=100 TO 400 STEP 100: Y1%=Y/256: Y2%=Y-256*Y1%: LPRINT CHR$(27);CHR$(
16);CHR$(Y1%);CHR$(Y2%);CHR$(140);: NEXT Y 'SMALL TICS
10099 RETURN

```

```

11000 '***** Y AXIS WITH STRING$ HASHMARKS *****
11010 LPRINT CHR$(18);STRING$(50,128);STRING$(201,131);STRING$(200,131);
401-DOT-LENGTH Y AXIS STARTING AT COLUMN 50
11020 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(50);CHR$(252);: FOR N=1 TO 4: LPRINT
STRING$(49,128);CHR$(140);STRING$(49,128);CHR$(252);: NEXT N
'LARGE AND SMALL HASHMARKS
11099 RETURN

```

```

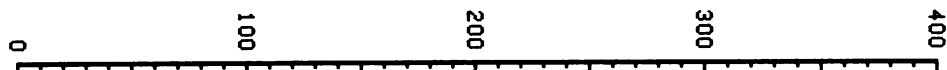
12000 '***** INTEGRATED Y AXIS AND HASHMARKS *****
12010 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(50);CHR$(255);: FOR N=1 TO
4: LPRINT STRING$(49,131);CHR$(143);STRING$(49,131);CHR$(255);: NEXT N
12099 RETURN

```

```

13000 '***** INTEGRATED AXIS WITH RULER-TYPE HASHMARKS *****
13010 HA$=STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(
143)+STRING$(9,131)+CHR$(143)+STRING$(9,131)
13020 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(50);CHR$(255);: FOR N=1 TO
4: LPRINT HA$;CHR$(191);HA$;CHR$(255);: NEXT N
13099 RETURN

```



```

15000 '***** Y AXIS WITH 5 X 7 PERPENDICULAR NUMBERING *****
15010 GOSUB 7000 'GET CODE SEQUENCES OF NUMBERS
15020 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(147);CHR$(18);N1$;STRING$(93,128);N2
$;STRING$(93,128);N3$;STRING$(93,128);N4$;CHR$(13); '1, 2, 3, 4" (1ST COLUMN O
F NUMBERS)
15030 FOR Y=147 TO 447 STEP 100: Y1%=Y/256: Y2%=Y-256*Y1%: LPRINT CHR$(27);CHR$(
16);CHR$(Y1%);CHR$(Y2%);N0$;: NEXT Y '0" (2ND COLUMN)
15035 LPRINT CHR$(13); 'LINE FEED (GRAPHIC)
15040 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(47);N0$;STRING$(93,128);N0$;STRING$(
93,128);N0$;STRING$(93,128);N0$;STRING$(93,128);N0$; '0" (3RD COLUMN)
15060 LPRINT CHR$(30);CHR$(13); 'LINE FEED (NON-GRAPHIC)
15070 GOSUB 13000 'Y AXIS WITH HASHMARKS
15099 RETURN

```

FIGURE 8-2. Some of the choices in programming top axes in horizontal graphs or Y axes in vertical graphs (TRS-80 codes).

want to draw both the axis and its hash marks without a line feed between them, a two- or three-dot thickness gives you more room to fit in the hash marks.

Notice that hash marks are positioned directly with a FOR-NEXT loop in the first example in figure 8-2, whereas in the second case only the bottom mark is positioned directly, and the higher ones are positioned by repeating a sequence of blank dot columns and a dot pattern. In the last two examples, the axis line and hash marks are not drawn in separate sweeps of the printhead, but instead they are done by repeating more complex sequences of dot codes. These last two are more efficient, but the second method is the fastest and quietest, and the first method, while easiest to program, is slow and seems to cause the greatest wear and tear on the printer.

The side axis and dashed vertical lines between the bars in figure 8-1's graph are handled by subroutine 2000. The programming involved here is nearly identical with that for putting hash marks on the top axis, and this is typical for grid lines that run parallel to the top axis in a horizontal graph or the Y axis in a vertical graph.

In adapting the BARGRAPH program for Apple printers, the same dot code 255 can be used for the bars themselves, but nearly all other dot codes will have to be converted and different control codes will be needed for line feeds, positioning, and graphic reservations (see table 3-1 and appendix C).

For TRS-80-type printers, `LPRINT CHR$(30);CHR$(27);CHR$(20);CHR$(18)` replaces the `27+"L"+N1+N2` sequence for entering graphic mode in the most condensed dot packing. `LPRINT CHR$(30);CHR$(27);CHR$(19);` needs to be inserted ahead of any printing of letters or numbers in standard 10-CPI. For the title at the top, this sequence needs to be followed by the control codes for entering and exiting the 5-CPI elongated pitch. The `CHR$(255)` dot code for the bar printing will, of course, be only seven dots wide instead of eight.

The 3-D bar style. If our first horizontal bar graph seems too much of a Plain Jane for your taste, you can try something with a little more flair, like figure 8-3's creation. Figure 8-3 shows a 3-D bar graph with some stylish labeling.

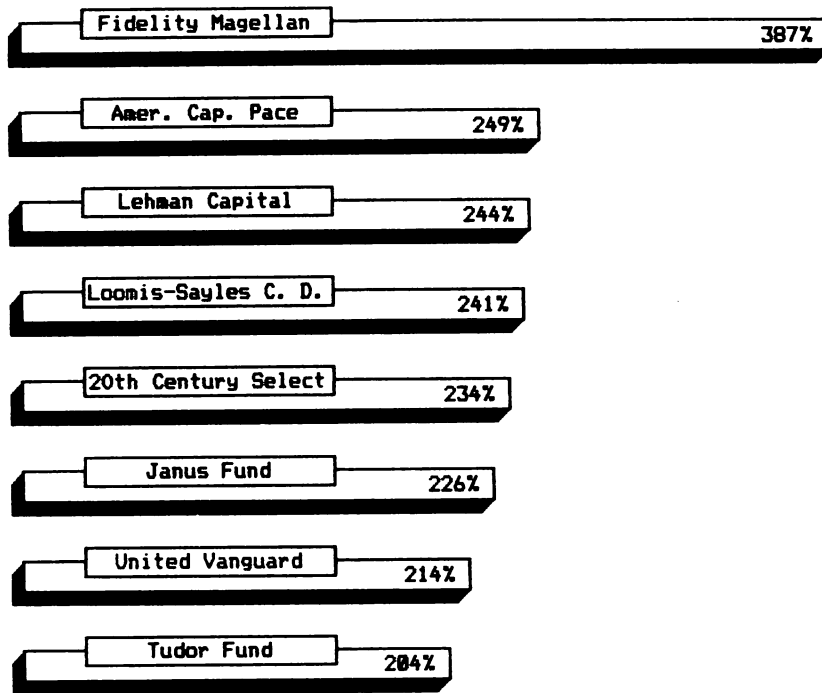
MUTUAL FUNDS: HIGHEST GAINERS, 1978-83

FIGURE 8-3. A 3-D horizontal bar graph from the BARSIN3D program for the DMP-400.

The program for this one is called BARSIN3D (program 8-2), written for IBM-Epson printers. It takes several lines just to draw a single bar in this program (lines 140-240), but they are worth the trouble, and a graph like this can get away with having no scaled axis (except perhaps in scientific journals). As a result, there is little more to the program than the bar drawing. Too bad they're not all this easy.

Program 8-3 is a version of BARSIN3D for TRS-80 printers.

PROGRAM 8-2. BARSIN3D/EPS. "Three-dimensional"
horizontal bar graph (IBM-Epson).

```

10 'PROGRAM 8-2: "BARSIN3D/EPS" -- THREE-DIMENSIONAL BAR GRAPH PROGRAM
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR IBM-EPSON PRINTERS AND TRS-80 COMPUTERS
30 CLEAR 500
40 LPRINT CHR$(27)"0";                                'MASTER RESET
50 LPRINT CHR$(27)"3"CHR$(10);                          '10/216" LINE FEED SET

```



```

100 LPRINT TAB(8);CHR$(27)"!";CHR$(52);"MUTUAL FUNDS: HIGHEST GAINERS, 1978-83";C
    HR$(27)"!";CHR$(8);: GOSUB 290: GOSUB 290
110 LPRINT CHR$(13);
120 READ N$,X: IF N$="END" THEN GOTO 270
130 X1=X-150
140 LPRINT CHR$(27)"K";CHR$(172)CHR$(0);STRING$(21,0);STRING$(30,1);CHR$(127);STR
    ING$(119,64);CHR$(127);CHR$(27)"K";CHR$(X1+1)CHR$(0);STRING$(X1+1,1);
150 LPRINT CHR$(13);
160 L=113-3*LEN(N$): LPRINT P0$;CHR$(27)"K";CHR$(L)CHR$(0);STRING$(L,0);N$;
170 LPRINT CHR$(13);
180 LPRINT CHR$(27)"K";CHR$(172)CHR$(0);STRING$(15,0);CHR$(3);CHR$(7);CHR$(15);CH
    R$(31);CHR$(63);CHR$(127);STRING$(30,0);CHR$(127);STRING$(119,1);CHR$(127);C
    HR$(27)"K";CHR$(X1+1)CHR$(0);STRING$(X1,0);CHR$(127);
190 LPRINT CHR$(13);
200 LPRINT CHR$(27)"K";CHR$(172)CHR$(0);STRING$(172,0);CHR$(27)"K";CHR$(X1-31)CHR$
    (0);STRING$(X1-31,0);CHR$(8);X;CHR$(8);"%";
210 LPRINT CHR$(13);
220 LPRINT CHR$(27)"K";CHR$(172)CHR$(0);STRING$(15,0);STRING$(6,127);STRING$(151,
    1);CHR$(27)"K";CHR$(X1+1)CHR$(0);STRING$(X1,1);CHR$(127);
230 GOSUB 290: LPRINT CHR$(27)"J";CHR$(11);CHR$(13);
240 LPRINT CHR$(27)"K";CHR$(172)CHR$(0);STRING$(15,0);STRING$(157,126);CHR$(27)"K
    "CHR$(X1)CHR$(0);STRING$(X1-5,126);CHR$(124);CHR$(120);CHR$(112);CHR$(96);CH
    R$(64);
250 LPRINT STRING$(3,13);                                'TRIPLE LINE FEED
260 GOTO 120
270 END
280 DATA Fidelity Magellan,387,Amer. Cap. Pace,249,Lehman Capital,244,Loomis-Say
    les C. D.,241,20th Century Select,234,Janus Fund,226,United Vanguard,214,Tud
    or Fund,204,END,999
290 'GRAPHIC (7/72") LINE FEED
300 LPRINT CHR$(27)"J";CHR$(21);: RETURN

```

PROGRAM 8-3. BARSIN3D/TRS. "Three-dimensional" horizontal bar graph (TRS-80).

```

10 'PROGRAM 8-3: "BARSIN3D/TRS" -- 3-D BAR GRAPH PROGRAM FOR TRS-80 PRINTERS
15 'Copyright (c) 1985 by John Warner Davenport
20 CLEAR 500
30 LPRINT CHR$(30);CHR$(27);CHR$(23);                'ELITE PRINTING
40 LPRINT CHR$(27);CHR$(31);                            'BOLDFACE PRINTING
50 P0$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)              'CARRIAGE RET. WITHOUT LINE FEED
60 LPRINT TAB(14);CHR$(15);"MUTUAL FUNDS: HIGHEST GAINERS, 1978-83";CHR$(14)
70 GOSUB 280: GOSUB 280
80 READ N$,X: IF N$="END" THEN GOTO 250
90 X1=X-150
100 LPRINT P0$;
110 LPRINT CHR$(18);STRING$(21,128);STRING$(30,192);CHR$(255);STRING$(119,129);C
    HR$(255);STRING$(X1+1,192);
120 GOSUB 270
130 L=106-3*LEN(N$): LPRINT P0$;STRING$(L,128);CHR$(30);N$;
140 GOSUB 270
150 LPRINT P0$;CHR$(18);STRING$(15,128);CHR$(224);CHR$(240);CHR$(248);CHR$(252);
    CHR$(254);CHR$(255);STRING$(30,128);CHR$(255);STRING$(119,192);CHR$(255);STR
    ING$(X1,128);CHR$(255);

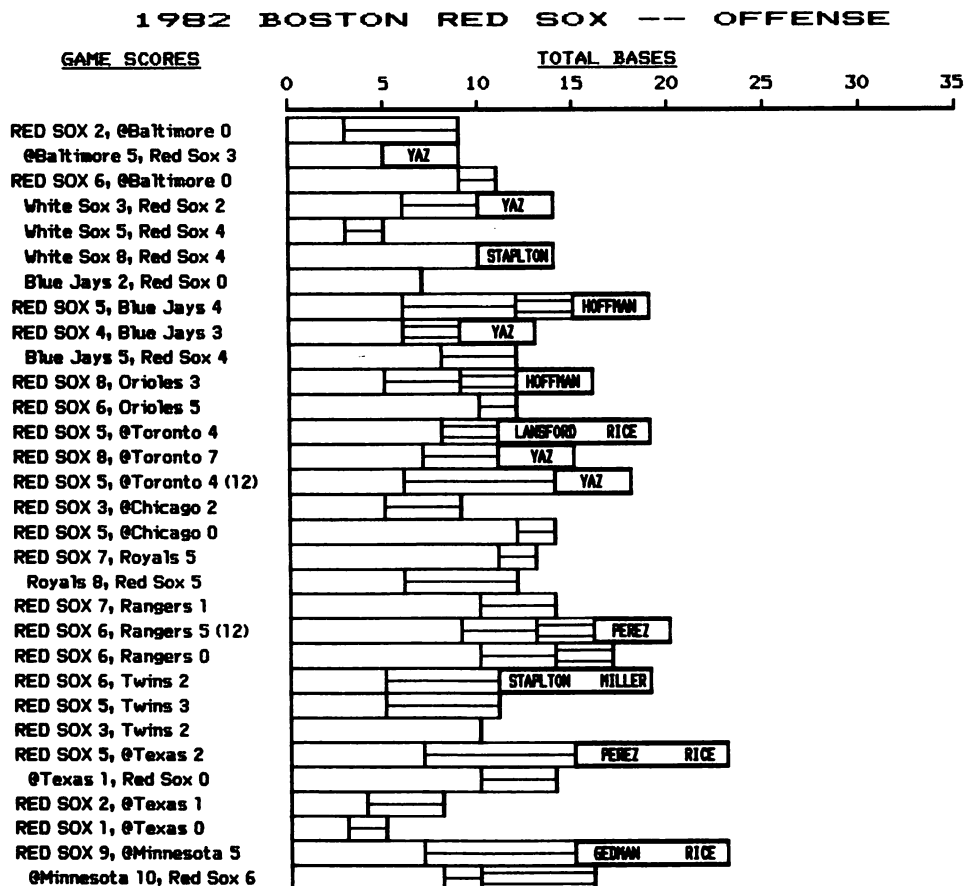
```

```

160 GOSUB 270
170 LPRINT P0$;STRING$(170,128);STRING$(X1-32,128);CHR$(30);X;CHR$(8);CHR$(11);"
    %";
180 PRINT X, X1, X1-32
190 GOSUB 270
200 LPRINT P0$;CHR$(18);STRING$(15,128);STRING$(6,255);STRING$(151,192);STRING$(
    X1,192);CHR$(255);
210 GOSUB 280
220 LPRINT P0$;STRING$(15,128);STRING$(156,191);STRING$(X1-5,191);CHR$(159);CHR$(
    143);CHR$(135);CHR$(131);CHR$(129);
230 GOSUB 280: GOSUB 280
240 GOTO 80
250 END
260 DATA Fidelity Magellan,387,Amer. Cap. Pace,249,Lehman Capital,244,Loomis-Say
    les C. D.,241,20th Century Select,234,Janus Fund,226,United Vanguard,214,Tud
    or Fund,204,END,999
270 FOR Z=1 TO 10: LPRINT CHR$(27);CHR$(51);: NEXT Z: RETURN '10/216" L. FEED
280 LPRINT CHR$(18);CHR$(13);: RETURN '7/72" LINE FEED

```

FIGURE 8-4. A segmented horizontal bar graph (SEGBARS program for the DMP-400).



Segmented bars. Next comes the segmented form of horizontal bar graph. In this form each bar's full length still represents a numerical total, but now the components that make up that total are shown in segments of the bar.

Figure 8-4 indicates some of the possibilities of the segmented horizontal bar graph. In this baseball example, "total bases" is defined just the way it is by the keepers of the official major-league records—the number of singles plus two times the number of doubles plus three times the number of triples plus four times the number of home runs. Here we're displaying a team's total-bases output in game-by-game fashion over the first three weeks of the season.

The hitting output that came in the form of singles is represented in the first (leftmost) segment, followed by the segments for doubles (if any) and triples (if any). These segments are distinguished from each other graphically by the number of horizontal lines that subdivide them. Each home run is distinguished by printing the name of the player who hit it, inside a four-unit rectangular space.

PROGRAM 8-4. SEGBARS. Segmented horizontal bar graph.

```

10 'PROGRAM 8-4: "SEGBARS" -- SEGMENTED HORIZONTAL BAR GRAPH
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 CLEAR 1000
40 P0$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)           ' POSITIONING STRING
50 BS$=CHR$(30)+CHR$(8)+CHR$(2)+CHR$(18)             ' ONE-DOT BACKSPACE
60 SP$=CHR$(18)+STRING$(33,128)+CHR$(30)              ' 33-DOT BLANK SPACE
70 LPRINT CHR$(30);CHR$(27);CHR$(32);                 ' RESET
80 N=9                                                  ' SCALE FACTOR
90 FOR Z=1 TO 2: LPRINT P0$;CHR$(58);CHR$(30);CHR$(27);CHR$(14);"1982 BOSTON RED
   SOX -- OFFENSE";CHR$(27);CHR$(15);: NEXT Z
100 LPRINT CHR$(18);STRING$(3,13);                     ' TRIPLE LINE FEED
110 LPRINT CHR$(30);CHR$(27);CHR$(31);                 ' BOLD PRINTING
120 LPRINT P0$;CHR$(24);CHR$(30);CHR$(15);"GAME SCORES";CHR$(14);P0$;CHR$(250);C
   HR$(30);CHR$(15);"TOTAL BASES";CHR$(14);CHR$(18);STRING$(2,13);
130 LPRINT P0$;CHR$(128);CHR$(30);"0";CHR$(18);STRING$(40,128);CHR$(30);"5";CHR$
   (18);STRING$(36,128);CHR$(30);"10";SP$;"15";SP$;"20";SP$;"25";SP$;"30";SP$;"
   35";CHR$(13);                                     ' NUMBERING OF AXIS
140 LPRINT P0$;CHR$(130);CHR$(255);
150 FOR Z=1 TO 7: LPRINT STRING$(44,224);CHR$(254);: NEXT Z
160 LPRINT CHR$(13);
170 READ O$,W,X,Y,Z,A$      ' OUTCOME, SINGLES, DOUBLES, TRIPLES, HOMERS, HITTERS
180 IF O$="END" THEN PRINT "PLOTING FINISHED": END
190 T=N*W+2*N*X+3*N*Y+4*N*Z      ' DEFINITION OF TOTAL BASES
200 S=N*W: D=S+2*N*X: R=D+3*N*Y: H=R+4*N*Z      ' SINGLE, DOUBLE, TRIPLE, HOME RUN
210 LPRINT P0$;CHR$(130);STRING$(T,136);      ' TOP EDGE OF BAR
220 LPRINT P0$;CHR$(130);CHR$(248);STRING$(S,128);BS$;CHR$(248);STRING$(D-S,128)

```

```

;BS$;CHR$(248);STRING$(R-D,128);BS$;CHR$(248);STRING$(H-R,152);BS$;CHR$(248)
;CHR$(13);
230 LPRINT CHR$(30);CHR$(27);CHR$(17);0$;CHR$(27);CHR$(19);P0$;CHR$(130);STRING$
(R+2,128);CHR$(30);CHR$(27);CHR$(20);A$;CHR$(27);CHR$(19);CHR$(18);
240 LPRINT P0$;CHR$(130);CHR$(255);STRING$(S,128);BS$;CHR$(255);STRING$(D-S,136)
;BS$;CHR$(255);STRING$(R-D,193);BS$;CHR$(255);STRING$(H-R,128);BS$;CHR$(255)
;CHR$(13);
250 LPRINT P0$;CHR$(130);CHR$(143);STRING$(S,128);BS$;CHR$(143);STRING$(D-S,128)
;BS$;CHR$(143);STRING$(R-D,128);BS$;CHR$(143);STRING$(H-R,140);BS$;CHR$(143)
;
260 LPRINT P0$;CHR$(130);CHR$(18);STRING$(T,136);      'BOTTOM EDGE OF BAR
270 GOTO 170
500 REM -- DATA FORMAT: SCORE (" , "),1B,2B,3B,HR," (NAME) "
510 REM *BOSTON RED SOX, 1982*
520 DATA "RED SOX 2, @Baltimore 0",3,3,0,0,"
530 DATA " @Baltimore 5, Red Sox 3",5,0,0,1," YAZ"
540 DATA "RED SOX 6, @Baltimore 0",9,1,0,0,"
550 DATA " White Sox 3, Red Sox 2",6,2,0,1," YAZ"
560 DATA " White Sox 5, Red Sox 4",3,1,0,0,"
570 DATA " White Sox 8, Red Sox 4",10,0,0,1," STAPLTON"
580 DATA " Blue Jays 2, Red Sox 0",7,0,0,0,"
590 DATA "RED SOX 5, Blue Jays 4",6,3,1,1," HOFFMAN"
600 DATA "RED SOX 4, Blue Jays 3",6,0,1,1," YAZ"
.
. (ETC.)
.

```

As program 8-4 (SEGBARS) shows, this segmented graph intermixes bit-image graphic elements with nongraphic characters, as with the first two graphs. Each bar is printed in four printhead passes by lines 210–260. Having the bars this fat allows room for printing a player's name inside a home-run rectangle and for spacing the scores of the games so that they are easy to read. This time the scale factor (N) is 9 (line 80), meaning that for every unit on the total-bases scale there are nine dot columns of horizontal space.

The tricky part of the programming is the segmenting of the bars (lines 220–250). Notice the repeated use of one-dot-column backspacing with the abbreviation BS\$ (defined in line 50). This precedes every vertical line that separates one segment from another, so that when zero values (of singles, doubles, triples, or home runs) occur, a vertical will be printed right on top of the preceding vertical instead of being one dot column to the right of it.

Two-way bar graphs. An important type of bar graph is the two-way horizontal form, in which bars fan out in both left

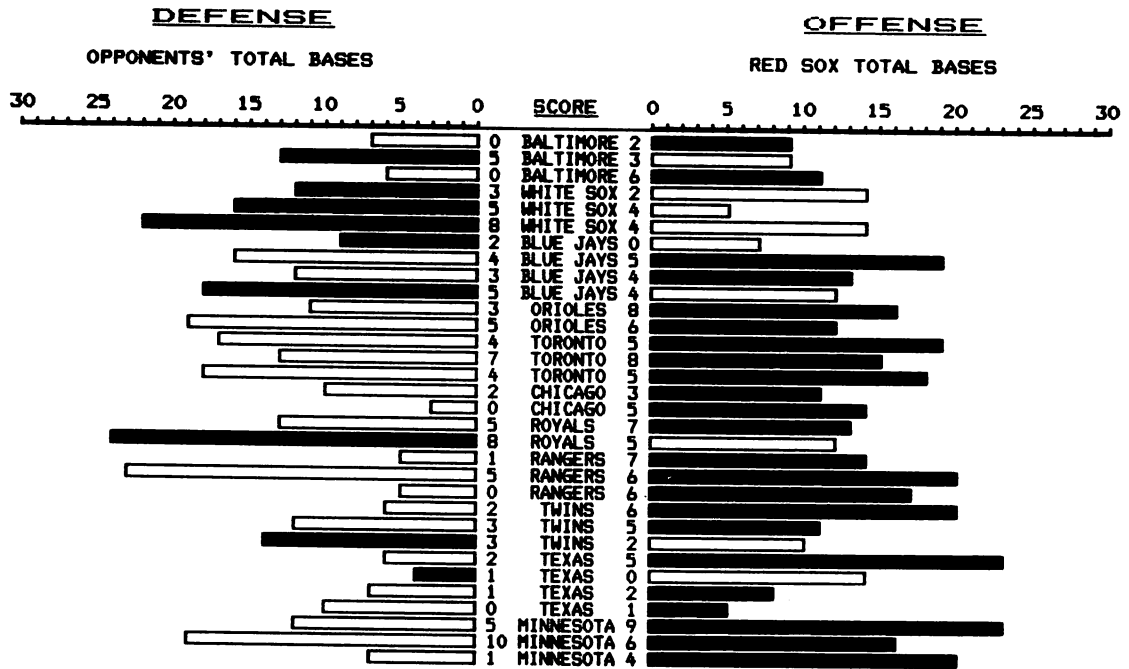


FIGURE 8-5. Expansion of figure 8-4's graph to a two-way horizontal graph (TWOWAYBG program for the DMP-400).

and right directions from a center column. This arrangement can be nifty for displaying power balances, interacting forces, paired comparisons, and other relationships.

We could get one example of a two-way bar graph if we just took the last one, moved the whole thing to the right, and added the Red Sox' opponents' total bases, but now with singles, doubles, triples, and homers running right to left from the (now-centered) list of game scores. Without a fifteen-inch printer like the DMP-400 or the FX-100, we'd be pretty cramped for space, although scales could be reduced for eight-inch printers. Baseball freaks have been known to do hand drawings of such two-way creations for whole 162-game seasons.

A simpler example of a two-way bar graph appears in figure 8-5. This one shows the same games played by the Boston Red Sox as in figure 8-4 and the same total-bases measure plotted in the same right-to-left fashion on the right side of the list of game scores. How many total bases the Red Sox

allowed their opponents in each of these games is shown on the left side. The format of the score is changed; the name of the opponent is centered in the central column, and on either side of the name the runs scored by the opponent (left) and the Red Sox (right) are given. This graph was plotted by program 8-5, TWOWAYBG.

This could be called an offense-defense graph. The main trick in the programming here is to get that set of defense (opponents') bars located properly. There's nothing very difficult about this, though. Each defensive total (the third item in each DATA statement) is simply subtracted from a constant (TBD, see line 190) value that represents the left edge of the central column. This tells the printer where to start printing a bar and to keep printing it until the left edge of the center column is reached. An offensive bar starts at the right edge of the center column and is simply printed according to the offensive total bases for the same game. Thus both bars for a game are printed with the printhead moving in its usual left-to-right direction, as is the score of the game in the center column, without any backspacing.

Another feature of the TWOWAYBG program highlights those few games the Red Sox lost, by having them swap "colors" with their successful opponents (i.e., the black bar goes to the winning team, and the empty bar goes to the loser, regardless of which team comes out ahead in total bases). In lines 220–250 the program decides who won each game (in the best possible way, by comparing the runs by the Red Sox with their opponents' runs) and then routes the bit-image dot code—CHR\$(255)—for shading the defensive or offensive bar accordingly.

To get an idea of some bigger jobs that can be handled by horizontal two-way bar graphs, study figure 8-6's floating two-way graph. This form unhooks the bars from a center column and lets them "float." And instead of standing for a single numerical value, each bar represents a total range of values. Within that range, a black square shows one selected performance that is highlighted. The usual purpose of such graphs is to provide a sweeping overview of how a pair of statistical measures have varied over a period of history.

Here is another example of "offense vs. defense," only now we're measuring offense in terms of runs scored (per

PROGRAM 8-5. TWOWAYBG. Bidirectional horizontal bar graph.

```

10 'PROGRAM 8-5: "TWOWAYBG" -- TWO-WAY HORIZONTAL BAR GRAPH
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS (USE BOOTHE'S DRIVER IF MODEL III)
30 CLEAR 1000
40 P0$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)           'POSITIONING STRING
50 P1$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(1)           '      "      "
60 SP$=CHR$(18)+STRING$(23,128)+CHR$(30)             '23-DOT BLANK SPACE
70 LPRINT P0$;CHR$(100);CHR$(30);CHR$(15);CHR$(27);CHR$(14);"DEFENSE";CHR$(14);C
   HR$(27);CHR$(15);
80 LPRINT P1$;CHR$(145);CHR$(30);CHR$(15);CHR$(27);CHR$(14);"OFFENSE";CHR$(14);C
   HR$(27);CHR$(15);STRING$(2,13);
90 LPRINT CHR$(27);CHR$(31);                         'BOLD PRINTING
100 LPRINT P0$;CHR$(70);CHR$(30);"OPPONENTS' TOTAL BASES";P1$;CHR$(120);CHR$(30)
    ;"RED SOX TOTAL BASES";STRING$(2,13);
110 LPRINT P0$;CHR$(35);CHR$(30);"30";SP$;"25";SP$;"20";SP$;"15";SP$;"10";CHR$(1
    8);STRING$(26,128);CHR$(30);"5";CHR$(18);STRING$(30,128);CHR$(30);"0";CHR$(1
    8);STRING$(23,128);CHR$(30);CHR$(15);"SCORE";CHR$(14);
120 LPRINT P1$;CHR$(73);CHR$(30);"0";CHR$(18);STRING$(29,128);CHR$(30);"5";CHR$(
    18);STRING$(26,128);CHR$(30);"10";SP$;"15";SP$;"20";SP$;"25";SP$;"30";CHR$(1
    3);
130 LPRINT CHR$(18);STRING$(40,128);STRING$(210,140);STRING$(81,136);STRING$(210
    ,140);
140 FOR Y=40 TO 253 STEP 7: LPRINT P0$;CHR$(Y);CHR$(143);: NEXT Y
150 FOR Y=331 TO 545 STEP 7: QY=FIX(Y/256): YY=FIX(Y-256*QY): LPRINT CHR$(27);CH
    R$(16);CHR$(QY);CHR$(YY);CHR$(143);: NEXT Y
160 LPRINT CHR$(13);                                   'CARRIAGE RETURN, LINE FEED
170 READ SC$,OF,DF           '(SCORE, OFFENSIVE TOTAL BASES, DEFENSIVE TOTAL BASES)
180 IF SC$="END" THEN END
190 OF%=7*OF: DF%=7*DF: TBD=250-DF%
200 REM -- PRINTING OF DEFENSE BAR (UNSHADED) AND SCORE
210 LPRINT P0$;CHR$(TBD);CHR$(255);STRING$(DF%,193);P0$;CHR$(250);CHR$(255);STR
    ING$(5,128);CHR$(30);CHR$(27);CHR$(23);SC$;CHR$(27);CHR$(19);
220 B=VAL(RIGHT$(SC$,2)): O=VAL(LEFT$(SC$,2))
230 REM -- NEXT TWO LINES BLACKEN THE BAR OF THE WINNING TEAM
240 IF B>O THEN LPRINT P1$;CHR$(76);STRING$(OF%,255);
250 IF O>B THEN LPRINT P0$;CHR$(TBD);STRING$(DF%,255);
260 REM -- PRINTING OF OFFENSE BAR
270 LPRINT P1$;CHR$(75);CHR$(255);STRING$(OF%,193);CHR$(255);           'PRINTING
    OF OFFENSE BAR
280 LPRINT CHR$(18);CHR$(10);CHR$(27);CHR$(50);CHR$(27);CHR$(50);           '9/72" LF
290 GOTO 170
500 DATA "0 BALTIMORE 2",9,7
510 DATA "5 BALTIMORE 3",9,13
520 DATA "0 BALTIMORE 6",11,6
530 DATA "3 WHITE SOX 2",14,12
540 DATA "5 WHITE SOX 4",5,16
550 DATA "8 WHITE SOX 4",14,22
560 DATA "2 BLUE JAYS 0",7,9
570 DATA "4 BLUE JAYS 5",19,16
580 DATA "3 BLUE JAYS 4",13,12
.
. (ETC.)
.

```

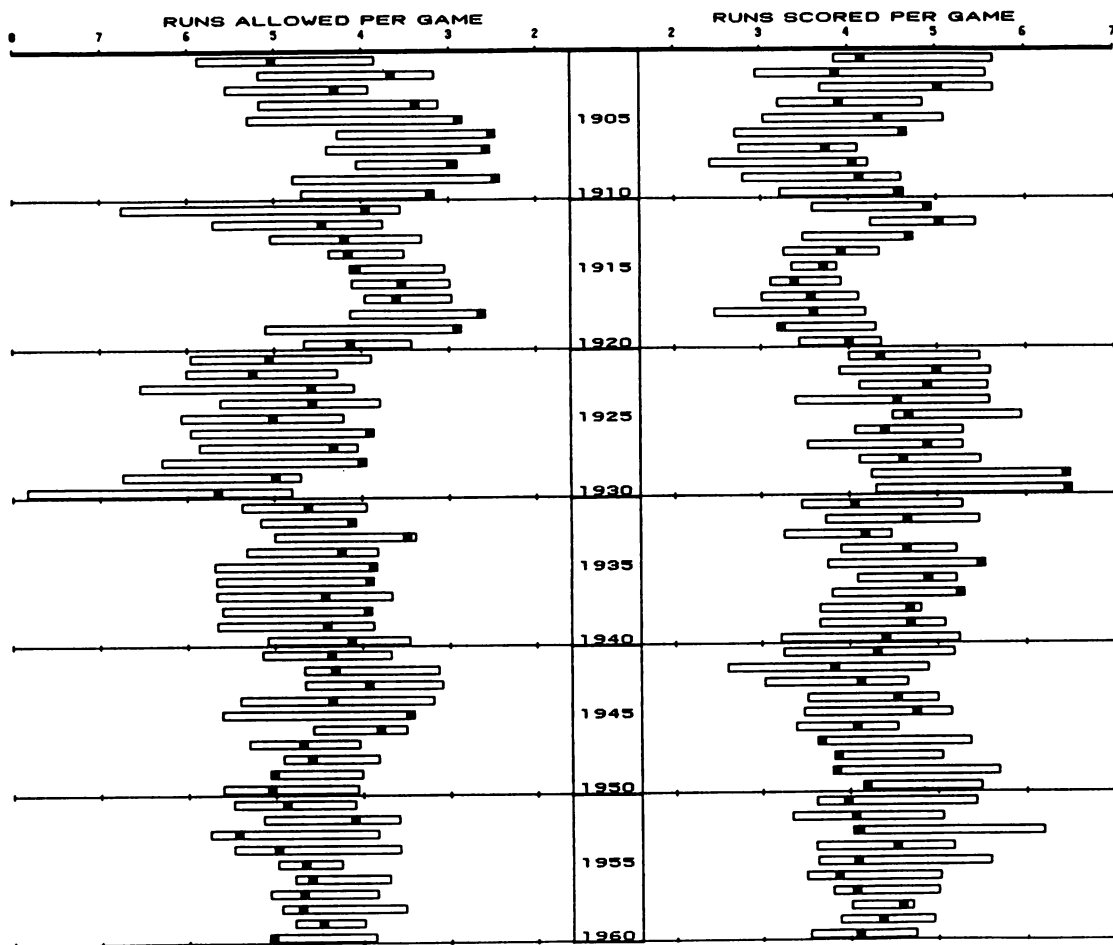


FIGURE 8-6. A floating two-way horizontal bar graph. The black squares indicate runs scored and runs allowed by the Chicago White Sox from 1901 to 1960. The bars show the ranges of the same measures for all American League teams in each year.

game) and defense by runs allowed (per game) in the history of American League baseball from 1901 through 1960. Each bar on the right side of the center column indicates the entire range of run scoring by all eight teams in the league, and on the left side the bars represent the whole range of the teams' performance in holding down their opponents' run scoring. The highlighted team represented by the black squares is the Chicago White Sox.

Except for having much more data to cope with, the program that produced figure 8-6 (not listed) doesn't differ greatly from the last two, and it is even more similar to those

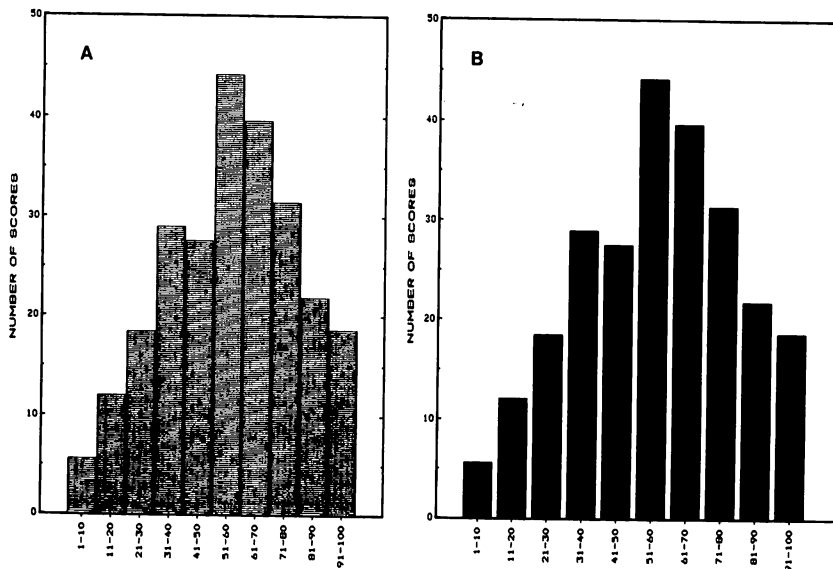
for the point-and-range time-series graphs in the next chapter.

Vertical Bar Graphs

Frequency distributions. To educators and social scientists there is probably no type of bar graph more familiar than the frequency histogram for displaying a frequency distribution. That is what we have in figure 8-7, which contains two histograms that were produced by program 8-6, FREQDIST.

By definition, a frequency distribution tells us the frequency of different scores along a measurement scale. So Frequency, Relative Frequency (where each frequency has been converted to a percentage of the total number of scores), or just plain Number of Cases will be the Y-axis label, and Score Values or Score Intervals (where scores have been grouped into narrow class intervals) will be the X-axis label. Program 8-6 takes a small or large number of scores in the range of 0 to 100, sorts them into ten-unit intervals (1-10, 11-20, . . . 91-100), and when all scores have been sorted, the number falling into each interval is counted. These pro-

FIGURE 8-7. A simple vertical bar graph, the frequency histogram, shown here in two different styles. Both examples were produced by the FREQDIST program for the DMP-400.



cesses are displayed on the computer's screen before any printing begins.

After the printing of the Y-axis label, NUMBER OF SCORES in this case, some homemade perpendicular numbers ranging from 0 to 50 are used to number the scale of the Y axis (subroutine 2000). Then the Y axis itself is printed (subroutines 5000 and 6000), with hash marks every five units on the frequency scale. The program goes on to start the X axes, top and bottom, which are lined up with the 50 and 0 points on the Y-axis scale (lines 5030 and 190).

PROGRAM 8-6. FREQDIST. Vertical bar graph (frequency histogram).

```

10 'PROGRAM 8-6: "FREQDIST" -- FREQUENCY DISTRIBUTION (HISTOGRAM)
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS (USE BOOTHE'S DRIVER IF MODEL III)
30 CLEAR 1000
40 BAR$=CHR$(128)+CHR$(255): T=128 '(FOR SOLID BARS, CHANGE BAR$ TO 'CHAR$(255
   )+CHR$(255)'; T = BLANK SPACE)
50 P$=CHR$(27)+CHR$(16) 'POSITIONING STRING
60 P0$=CHR$(27)+CHR$(16)+CHR$(0) ' " " "
70 FOR N=1 TO 2: LPRINT CHR$(30);CHR$(27);CHR$(19);CHR$(27);CHR$(32);P$;CHR$(1);
   CHR$(20);CHR$(27);CHR$(14);CHR$(27);CHR$(17);"NUMBER OF SCORES";CHR$(27);CHR
   $(19);CHR$(27);CHR$(15);: NEXT N
80 LPRINT CHR$(27);CHR$(31); 'BOLD PRINTING
90 GOSUB 8000: GOSUB 8000: GOSUB 8000 'TRIPLE LINE FEED
100 UL=600: GOSUB 2010 'Y-AXIS NUMBERING
110 GOSUB 8000
120 LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50); '2/72" LINE FEED
130 A=4: B=2: C=1 'SPACING OF AXES & BARS, BAR WIDTH
140 GOSUB 1030 'FOR VALUES OF QU, UU
150 GOSUB 6010 'FOR Y-AXIS
160 GOSUB 5010 'FOR Y-AXIS HASHMARKS
170 FOR X=1 TO A
180 LPRINT CHR$(30);CHR$(27);CHR$(32); 'END BOLD PRINTING
190 FOR Y=99 TO UL STEP UL-99: GOSUB 1010: LPRINT P$;CHR$(QY);CHR$(YY);CHR$(18)
   ;STRING$(2,255);: NEXT Y 'BLANK X-AXIS SPACING
200 GOSUB 8000
210 NEXT X
220 READ L$,M
230 IF L$="END" THEN GOTO 370
240 GOSUB 3000 'PRINT LEFT EDGE OF BAR
250 FOR W=1 TO B
260 GOSUB 1020: LPRINT CHR$(18);P0$;CHR$(99);STRING$(2,255);: FOR Z=1 TO 5*M: L
   PRINT BAR$;: NEXT Z: LPRINT P$;CHR$(QM);CHR$(MM);CHR$(T);P$;CHR$(QU);CHR$(UU)
   ;STRING$(2,255);CHR$(13); 'PRINT LEFT HALF OF BAR
270 NEXT W
280 PRINT L$,M,QM,MM
290 LPRINT CHR$(30);TAB(9);CHR$(27);CHR$(20);CHR$(27);CHR$(14);L$;CHR$(27);CHR$(
   15);CHR$(27);CHR$(19);

```

```

300 LPRINT CHR$(18);P0$;CHR$(96);STRING$(4,136);
310 FOR W=1 TO B+1: LPRINT P0$;CHR$(99);STRING$(2,255);: FOR Z=1 TO 5*M: LPRINT
    BAR$;: NEXT Z: LPRINT P$;CHR$(QM);CHR$(MM);CHR$(T);P$;CHR$(QU);CHR$(UU);STR
    NG$(2,255);CHR$(13);: NEXT W          'PRINT RIGHT HALF OF BAR
320 GOSUB 4000                                'PRINT RIGHT EDGE OF BAR
330 FOR Z=1 TO C+1
340 LPRINT P0$;CHR$(99);STRING$(2,255);P$;CHR$(QU);CHR$(UU);STRING$(2,255);CHR$(
    13);                                'X-AXIS SPACE BETWEEN BARS
350 NEXT Z
360 GOTO 220
370 FOR X=1 TO A: GOSUB 1030 : LPRINT CHR$(18);P0$;CHR$(99);STRING$(2,255);P$;CH
    R$(QU);CHR$(UU);STRING$(2,255);CHR$(13);: NEXT X
380 GOSUB 7000
390 FOR Y=200 TO UL-100 STEP 100: GOSUB 1010 : LPRINT P$;CHR$(QY);CHR$(YY);CHR$(
    18);CHR$(143);: NEXT Y          'LONG HASHMARKS
400 FOR Y=150 TO UL-50 STEP 100: GOSUB 1010 : LPRINT P$;CHR$(QY);CHR$(YY);CHR$(1
    8);CHR$(140);: NEXT Y          'SHORT HASHMARKS
410 LPRINT P0$;CHR$(99);STRING$(2,191);P$;CHR$(QU);CHR$(UU);STRING$(2,191);
    'END OF X-AXES (BOTTOM & TOP)
420 END
500 DATA " 1-10",5.68
510 DATA " 11-20",12.137
520 DATA " 21-30",18.54
530 DATA " 31-40",29.08
540 DATA " 41-50",27.66
550 DATA " 51-60",44.25
560 DATA " 61-70",39.73
570 DATA " 71-80",31.47
580 DATA " 81-90",21.88
590 DATA " 91-100",18.76
600 DATA END,-99
990 END
1000 '***** CONVERSIONS *****
1010 QY=FIX(Y/256): YY=FIX(Y-256*QY): RETURN
1020 QM=FIX((10*M+100)/256): MM=FIX((10*M+100)-256*QM): RETURN
1030 QU=FIX(UL/256): UU=FIX(UL-256*QU): RETURN
2000 '***** Y-AXIS, NUMBERING *****
2010 N1$=CHR$(156)+STRING$(4,136)+CHR$(140)+CHR$(136): LPRINT CHR$(18);P0$;CHR$(
    197);N1$;                                '"1"
2020 LPRINT CHR$(18);P$;CHR$(1);CHR$(41);CHR$(190);CHR$(130);CHR$(132);CHR$(152)
    ;CHR$(160);CHR$(162);CHR$(156);                                '"2"
2030 LPRINT CHR$(18);P$;CHR$(1);CHR$(141);CHR$(156);CHR$(162);CHR$(160);CHR$(152)
    ;CHR$(160);CHR$(162);CHR$(156);                                '"3"
2040 LPRINT CHR$(18);P$;CHR$(1);CHR$(241);CHR$(144);CHR$(144);CHR$(190);CHR$(148)
    ;CHR$(152);CHR$(144);CHR$(144);                                '"4"
2050 LPRINT CHR$(18);P$;CHR$(2);CHR$(85);CHR$(156);CHR$(162);CHR$(160);CHR$(160)
    ;CHR$(158);CHR$(130);CHR$(190);                                '"5"
2060 LPRINT CHR$(18);CHR$(10);                                'LINE FEED
2070 FOR Y=97 TO UL-3 STEP 100: GOSUB 1010 : LPRINT CHR$(18);P$;CHR$(QY);CHR$(YY
    );CHR$(156);CHR$(28);CHR$(5);CHR$(162);CHR$(156);: NEXT Y          '"0"
2080 RETURN
3000 '***** PRINTING OF BAR'S LEFT EDGE *****
3010 LPRINT CHR$(18);P0$;CHR$(100);
3020 FOR Z=1 TO 10*M: LPRINT CHR$(129);: NEXT Z

```

```

3030 RETURN
4000 '***** PRINTING OF BAR'S RIGHT EDGE *****
4010 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(19);: FOR
      V=1 TO 5: LPRINT CHR$(27);CHR$(50);: NEXT V      '7/72" REVERSE FEED
4020 LPRINT CHR$(18);P0$;CHR$(100);
4030 FOR Z=1 TO 10*M: LPRINT CHR$(192);: NEXT Z
4040 RETURN
5000 '***** Y-AXIS HASHMARKS *****
5010 FOR Y=200 TO UL-100 STEP 100: GOSUB 1010 : LPRINT P$;CHR$(QY);CHR$(YY);CHR
      $(18);CHR$(248);: NEXT Y      'LONG HASHMARKS
5020 FOR Y=150 TO UL-50 STEP 100: GOSUB 1010 : LPRINT P$;CHR$(QY);CHR$(YY);CHR$(
      18);CHR$(152);: NEXT Y      'SHORT HASHMARKS
5030 FOR Y=UL TO 99 STEP -(UL-99): GOSUB 1010 : LPRINT P$;CHR$(QY);CHR$(YY);CHR$
      (18);CHR$(254);CHR$(254);: NEXT Y      'BEGINNING OF X-AXES (BOTTOM & TOP)
5040 GOSUB 8000
5050 RETURN
6000 '***** LEFT Y AXIS *****
6010 LPRINT CHR$(18);P0$;CHR$(99);STRING$(201,134);STRING$(100,134);STRING$(200,
      134);: RETURN
7000 '***** RIGHT Y AXIS *****
7010 LPRINT CHR$(18);P0$;CHR$(99);STRING$(201,176);STRING$(100,176);STRING$(200,
      176);: RETURN
8000 LPRINT CHR$(18);CHR$(13);: RETURN      'GRAPHIC LINE FEED

```

The amount of blank X-axis space between the left Y axis and the first bar is controlled by the value of A in line 130. That same line sets the width of the bars (B) and the spacing between the bars (C). The plotting of each bar is done in a number of printhead sweeps that is equal to B (lines 250–320 and subroutines 3000 and 4000).

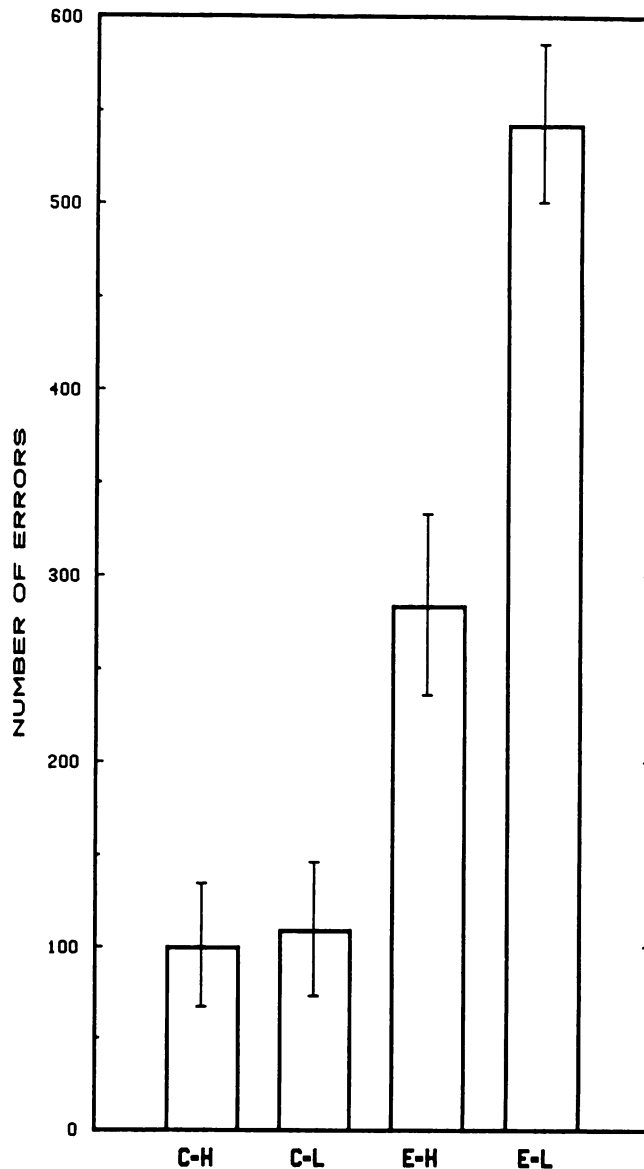
Built-in (upright) numbers are used for numbering the score intervals (1–10, 11–20, etc.), but this is the easy way out of the X-axis labeling problem. Purists may object to numbers lying at right angles to the X axis, even though this is commonly done with frequency histograms drawn by hand. Some later graphs and figure 1-3 (chapter 1) show more professional-looking alternatives for numbering and labeling the X axis.

Means with standard errors. Next comes a familiar favorite in the scientific journals—the mean-and-standard-error bar graph. For this one, we have program 8-7 (MSEGRAPH) and figure 8-8 for its printout. Now, instead of displaying frequencies (on Y) of score values (on X), we have a performance measure of some sort on the Y axis and two or more groups of performers (patients, rats, student volunteers, etc.) identified along the X axis. The lengths of the bars tell how

the groups have come out in their mean (arithmetic average) performances.

Above and below the top of each bar, vertical lines give the size of the standard error associated with the group's mean performance value. A standard error's size varies with

FIGURE 8-8. The mean-and-standard-error bar graph, widely used in scientific journals (MSEGRAPH program for the DMP-400).



both the amount of variation across individuals' scores within a group and the size of the group (number of scores); the smaller the variation and the larger the group, the smaller the standard error will be.

PROGRAM 8-7. MSEGRAPH. Mean-and-standard-error bar graph.

```

10 'PROGRAM 8-7: "MSEGRAPH" -- BAR GRAPH OF MEANS & STANDARD ERRORS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTER (USE BOOTHE'S DRIVER IF MODEL III)
30 CLEAR 1000
40 GOSUB 8000
50 PS$=CHR$(18)+CHR$(27)+CHR$(16)
60 P0$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)
70 P1$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(1)
80 P2$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(2)
90 P3$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(3)
100 PX$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(100)
110 UL=700: A=6: B=3: C=4
120 T$="NUMBER OF ERRORS"
130 Y=330-LEN(T$): GOSUB 600
140 FOR Z=1 TO 2
150 LPRINT CHR$(30);CHR$(27);CHR$(32);CHR$(27);CHR$(19);CHR$(27);CHR$(16);CHR$(Q
Y);CHR$(YY);CHR$(27);CHR$(14);CHR$(27);CHR$(17);"NUMBER OF ERRORS";CHR$(27);
CHR$(19);CHR$(27);CHR$(15);
160 NEXT Z
170 LPRINT CHR$(10)
180 LPRINT CHR$(30);CHR$(27);CHR$(31);
190 GOSUB 1000
200 GOSUB 6020
210 LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50);
220 REM -- PLOTTING OF Y AXIS
230 GOSUB 690
240 GOSUB 4000
250 GOSUB 3010
260 FOR Z=1 TO A: GOSUB 690: GOSUB 4050: NEXT Z
270 READ L$,M,SE
280 IF L$="END" THEN GOTO 390
290 GOSUB 2000
300 GOSUB 630
310 FOR W=1 TO B: GOSUB 2030: NEXT W
320 GOSUB 5000: GOSUB 660: LPRINT PS$;CHR$(QS);CHR$(SS);CHR$(255);STRING$(2*SE,1
36);CHR$(255);
330 FOR W=1 TO B+1: GOSUB 2030: NEXT W
340 GOSUB 2000
350 FOR Z=1 TO C
360 GOSUB 690: GOSUB 4050
370 NEXT Z
380 GOTO 270
390 FOR Z=1 TO 2: GOSUB 690: GOSUB 4050: NEXT Z
400 GOSUB 4030: GOSUB 3060
410 END
500 '***** DATA (LABEL, MEAN, STANDARD ERROR) *****
510 DATA CONTROL-HIGH,100,33

```

```

520 DATA CONTROL-LOW,109,36
530 DATA EXPERIMENTAL-HIGH,284,48
540 DATA EXPERIMENTAL-LOW,543,42
550 DATA END,-1,-1
600 '***** POSITION CONVERSION FOR Y *****
610 QY=FIX(Y/256): YY=FIX(Y-256*QY)
620 RETURN
630 '***** POSITION CONVERSION FOR M (MEAN) *****
640 QM=FIX((M+100)/256): MM=FIX((M+100)-256*QM)
650 RETURN
660 '***** POSITION CONVERSION FOR V (M-SE) *****
670 V=M-SE: QS=FIX((V+100)/256): SS=FIX((V+100)-256*QS)
680 RETURN
690 '**** POSITION CONVERSION FOR UL (UPPER LIMIT OF Y AXIS) ****
700 QU=FIX(UL/256): UU=FIX(UL-256*QU)
710 RETURN
1000 '***** NUMBERING OF Y AXIS *****
1010 LPRINT P0$;CHR$(197);N1$;     '"1"
1020 LPRINT P1$;CHR$(41);N2$;     '"2"
1030 LPRINT P1$;CHR$(141);N3$;     '"3"
1040 LPRINT P1$;CHR$(241);N4$;     '"4"
1050 IF UL<500 THEN GOTO 1090
1060 LPRINT P2$;CHR$(85);N5$;     '"5"
1070 IF UL<600 THEN GOTO 1090
1080 LPRINT P2$;CHR$(185);N6$;     '"6"
1090 LPRINT CHR$(10);
1100 FOR Y=197 TO UL-3 STEP 100: GOSUB 600 : LPRINT PS$;CHR$(QY);CHR$(YY);N0$;:
      NEXT Y: GOSUB 6020     '"0"
1110 FOR Y=97 TO UL-3 STEP 100: GOSUB 600 : LPRINT PS$;CHR$(QY);CHR$(YY);N0$;: N
      EXT Y: GOSUB 6020     '"0"
1120 RETURN
2000 '***** PRINTING OF BAR SIDES *****
2010 LPRINT PX$;: FOR Z=1 TO M+1: LPRINT CHR$(131);: NEXT Z
2020 RETURN
2030 '***** PRINTING OF BAR TOPS *****
2040 LPRINT PS$;CHR$(QM);CHR$(MM);CHR$(30);CHR$(8);CHR$(2);CHR$(18);STRING$(2,25
      5);
2050 GOSUB 4050
2060 RETURN
3000 '***** Y-AXIS HASHMARKS *****
3010 FOR Y=200 TO UL-100 STEP 100: GOSUB 600: LPRINT PS$;CHR$(QY);CHR$(YY);CHR$
      (248);: NEXT Y     'LONG HASHMARKS
3020 FOR Y=150 TO UL-50 STEP 100: GOSUB 600: LPRINT PS$;CHR$(QY);CHR$(YY);CHR$(1
      52);: NEXT Y     'SHORT HASHMARKS
3030 FOR Y=UL TO 99 STEP -(UL-99): GOSUB 600: LPRINT PS$;CHR$(QY);CHR$(YY);STRIN
      G$(2,254);: NEXT Y     'BEGINNING OF X-AXES (BOTTOM & TOP)
3040 GOSUB 6020
3050 RETURN
3060 FOR Y=200 TO UL-100 STEP 100: GOSUB 600: LPRINT PS$;CHR$(QY);CHR$(YY);CHR$(
      143);: NEXT Y     'LONG HASHMARKS
3070 FOR Y=150 TO UL-50 STEP 100: GOSUB 600: LPRINT PS$;CHR$(QY);CHR$(YY);CHR$(
      140);: NEXT Y     'SHORT HASHMARKS
3080 LPRINT P0$;CHR$(99);STRING$(2,191);PS$;CHR$(QU);CHR$(UU);STRING$(2,191)
3090 RETURN

```

```

4000 '***** Y AXES *****
4010 LPRINT PX$;: FOR Y=100 TO UL: LPRINT CHR$(134);: NEXT Y
4020 RETURN
4030 LPRINT PX$;: FOR Y=100 TO UL: LPRINT CHR$(176);: NEXT Y
4040 RETURN
4050 '***** X-AXIS SEGMENTS (LOWER & UPPER) *****
4060 LPRINT P0$;CHR$(99);STRING$(2,255);PS$;CHR$(QU);CHR$(UU);STRING$(2,255);CHR
$(10);
4070 RETURN
5000 '***** LABELING OF X AXIS *****
5010 GOSUB 7000 'GET CAPITAL LETTER STRINGS
5020 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(19);
5030 PL$=CHR$(13)+CHR$(27)+CHR$(50)+P0$+CHR$(80)
5040 IF L$="CONTROL-LOW" THEN LPRINT P0$;CHR$(80);CC$;PL$;HY$;PL$;CL$;: GOSUB 60
10 'C-L"
5050 IF L$="CONTROL-HIGH" THEN LPRINT P0$;CHR$(80);CC$;PL$;HY$;PL$;CH$;: GOSUB 6
010 'C-H"
5060 IF L$="EXPERIMENTAL-LOW" THEN LPRINT P0$;CHR$(80);CE$;PL$;HY$;PL$;CL$;: GOS
UB 6010 'E-L"
5070 IF L$="EXPERIMENTAL-HIGH" THEN LPRINT P0$;CHR$(80);CE$;PL$;HY$;PL$;CH$;: GO
SUB 6010 'E-H"
5080 FOR Z=1 TO 3: LPRINT CHR$(27);CHR$(50);: NEXT Z
5090 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(27);CHR$(
30);CHR$(19);
5100 RETURN
6000 '***** LINE FEEDS *****
6010 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(56);CHR$(19);: RETURN
6020 LPRINT CHR$(18);CHR$(10);: RETURN
7000 '**** BOLDFACE CAPITAL LETTERS FOR X-AXIS LABELING ****
7010 CC$=CHR$(18)+CHR$(190)+CHR$(255)+CHR$(227)+STRING$(4,131)+CHR$(227)+CHR$(25
5)+CHR$(190) 'C"
7020 CE$=CHR$(18)+STRING$(2,255)+STRING$(2,131)+STRING$(2,159)+STRING$(2,131)+ST
RING$(2,255) 'E"
7030 CH$=CHR$(18)+STRING$(4,227)+STRING$(2,255)+STRING$(4,227) 'H"
7040 CL$=CHR$(18)+STRING$(2,255)+STRING$(8,131) 'L"
7050 HY$=CHR$(18)+STRING$(3,128)+STRING$(3,190)+STRING$(4,128) 'HYPHEN
7060 RETURN
8000 '***** 5 X 11 CONDENSED NUMBERS FOR Y-AXIS NUMBERING *****
8010 CD$=CHR$(30)+CHR$(27)+CHR$(20)+CHR$(18): NM$=CHR$(30)+CHR$(27)+CHR$(19)+CHR
$(18)
8020 N1$=CD$+CHR$(156)+STRING$(7,136)+CHR$(140)+STRING$(2,136)+NM$
8030 N2$=CD$+CHR$(190)+STRING$(2,130)+CHR$(132)+CHR$(136)+CHR$(144)+STRING$(2,16
0)+STRING$(2,162)+CHR$(156)+NM$
8040 N3$=CD$+CHR$(156)+STRING$(2,162)+CHR$(160)+CHR$(144)+CHR$(156)+CHR$(144)+CH
R$(160)+STRING$(2,162)+CHR$(156)+NM$
8050 N4$=CD$+STRING$(4,144)+CHR$(190)+CHR$(146)+CHR$(148)+CHR$(152)+STRING$(3,14
4)+NM$
8060 N5$=CD$+CHR$(156)+STRING$(2,162)+STRING$(3,160)+CHR$(158)+STRING$(3,130)+CH
R$(190)+NM$
8070 N6$=CD$+CHR$(156)+STRING$(4,162)+CHR$(158)+STRING$(2,130)+STRING$(2,162)+CH
R$(156)+NM$
8080 N0$=CD$+CHR$(156)+STRING$(9,162)+CHR$(156)+NM$
8090 HM$=CD$+STRING$(4,128)+STRING$(3,190)+STRING$(4,128)+NM$
8100 RETURN

```


The main reason for wanting to know the standard errors of means as well as the means themselves is to be better able to judge whether differences among groups' means could have occurred by chance. If the standard errors above and below one group's mean show no overlap with another group's mean, plus or minus one standard error, then it is quite unlikely that the difference between the groups' means arose from mere chance variations, and we are justified in saying that the groups are really different in their performances as a result of different abilities, other characteristics, or experimental treatments they have received. In the case of the groups in figure 8-7, the standard errors in general are small enough to permit us to conclude that both experimental (E-H and E-L) groups made significantly more errors than both control (C-H and C-L) groups, and one of the experimental groups (E-H) scored significantly better (made fewer errors) than the other (E-L).

Compared to the FREQDIST program (program 8-6), the main new element in the MSEGRAPH program is the drawing of the standard-error verticals. The arithmetic for calculating the dot-column positions of these verticals is done in subroutine 660, and the commands for the drawing of the verticals are in line 320, halfway through the drawing of each bar.

The hard part is the labeling of the axes. Over a third of the program (subroutines 690, 1000, 3000, 4000, and 8000) is concerned with the Y axis, and a fifth of the program (subroutines 5000 and 7000) is devoted to the X-axis labeling. These X-axis subroutines are called (in line 320) every time the plotting reaches the midpoint of a bar. For each label, seven-by-ten bold perpendicular capital letters (C, E, H, and L) and a hyphen (HY\$) are defined in subroutine 7000 and printed by subroutine 5000. The printing is done as in the WRDCRAFT program for spelling of perpendiculars (program 7-2), with each character in the label requiring a positioning code and line feed. But now, in order to get the label in the proper position on the graph, its printing has to be preceded and followed by reverse line feeds (these are commanded in lines 5020 and 5090).

Variations of the MSEGRAPH program can provide grouped vertical bar graphs, such as the one in chapter 1 (figure 1-3). With bars in clusters, there is more need for

visual contrasts in the bars. White, 50% shaded, and black bars make for a good contrast in groups of three, and other kinds of shading can be used for larger groups. For white (unfilled) bars, the drawing would proceed the same as for all bars in the FREQDIST and original MSEGGRAPH program. For shaded bars, the space between the lower X axis and the top of the bar would have to be “painted” in a series of printhead passes for the whole width of the bar, in combination with the usual printing of the bar’s top and the X axes. Shading patterns inside FOR-NEXT loops, like those in the examples of “statistical tapes” (figure 6-2), can be used for the bar painting.

Standard errors can still be displayed as usual for white and shaded bars. For black bars, showing the “- 1 standard error” (below the top of the bar) merely involves a black-white reversal in the printing of the middle part of the bar. On the printhead sweep that prints the standard error verticals, the program line changes from

```
320 GOSUB 5000: GOSUB 660: LPRINT PS$;
    CHR$(QS);CHR$(SS);CHR$(255);
    STRING$(2*SE,136);CHR$(255);
```

to

```
320 GOSUB 5000: GOSUB 660: LPRINT PX$;
322 FOR N=1 TO V: LPRINT CHR$(255);: NEXT N
324 LPRINT CHR$(128);      'BLANK
326 FOR N=1 TO SE-1: LPRINT CHR$(247);:
    NEXT N
328 FOR N=1 TO SE-1: LPRINT CHR$(136);:
    NEXT N
329 LPRINT CHR$(255);
```

The important thing to grasp here is that the lower half of the standard-error vertical is printed by CHR\$(247), whose dot pattern consists of three dots, a blank, and three more dots. In combination with line 324’s CHR\$(128) and the preceding and following print lines printed all black, this yields a white-on-black vertical from the top of the bar down to the - 1 SE point.

The labeling of the grouped bar graph in figure 1-3 involved not only labels on the X axis under each bar (A, B, C, and D, repeated over the groups of bars) but, underneath those, labels (Before, During, and After) for the groupings as well. The former were positioned in the same fashion as the bar labels in figure 8-8, except that no reverse line feeds were needed because each letter could be printed in a single pass of the printhead. The latter were printed by interrupting the plotting of the graph at the midpoint of each *group* of bars instead of the midpoint of each bar. With four bars in each grouping, this meant interrupting between the second and third bars of each group, and (as in figure 8-8) having reverse line feeds occur before and after the printing of each of three labels.

Vertical floating-bar graphs. The bars in vertical floating-bar graphs are not anchored to the X axis of any other horizontal reference line. They are free to “hang in midair” in order to represent the highest and lowest values of any measure, and thereby also show the range of values. For a simple example, see figure 8-9’s weather graph, which shows the daily high and low temperatures for a month in Madison, Wisconsin. This graph differs from the one appearing every month in Madison’s *Wisconsin State Journal* only in the kind of shading or stippling that fills each bar and the amount of verbiage between the Y axes.

The TEMPBARS program that produced it (program 8-8) is similar to programs for anchored bars except for starting the printing of a bar above the X axis, at the level corresponding to each “lowest” value on the Y axis, instead of starting at the X axis itself. The positioning of each bar is more like the positioning of the standard-error verticals than the positioning of the bars in the MSEGGRAPH examples. The shading of the bars is done by repetition of two spaced-dot codes, CHR\$(213) and CHR\$(170).

We also have here another example of different labeling of the left and right Y axes. Besides the different numbering of the Fahrenheit and centigrade scales, the lettering for the latter (DEGREES CENTIGRADE) uses homemade upside-down characters (subroutine 8000) which have to be spelled backwards (see line 8130).

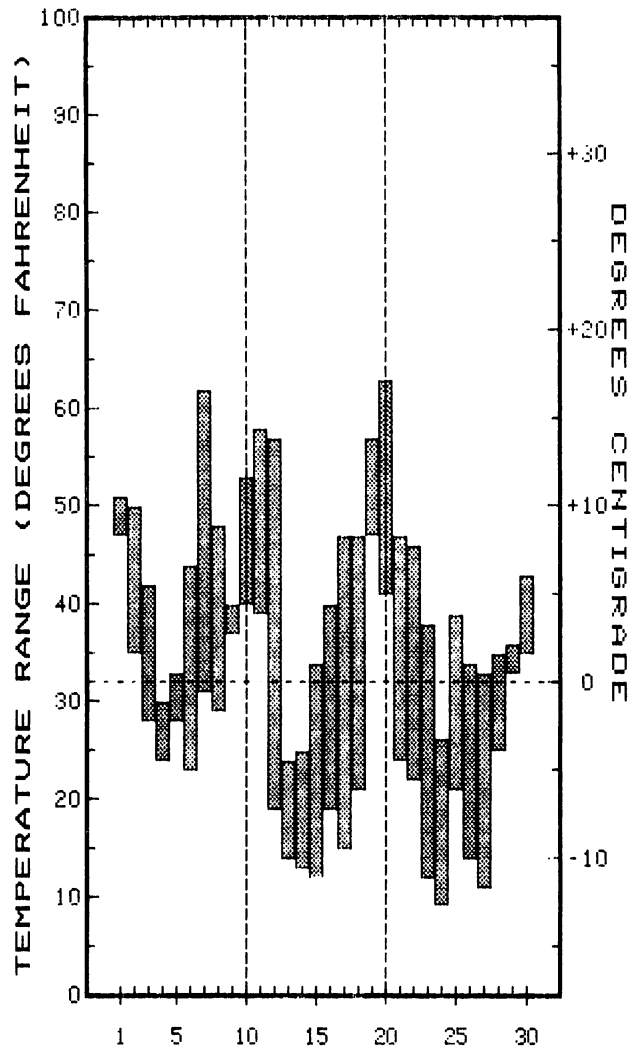


FIGURE 8-9. A vertical floating-bar graph (TEMPBARS program for the DMP-400).

PROGRAM 8-8. TEMPBARS. Floating vertical bar graph.

```

10 'PROGRAM 8-8: "TEMPBARS" -- FLOATING BAR GRAPH -- DAILY TEMPERATURE RANGES
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS (USE BOOTHE'S DRIVER WITH MODEL III)
30 CLEAR 2000
100 RLF$=CHR$(30)+CHR$(20)+CHR$(27)+CHR$(30)+CHR$(27)+CHR$(30)+CHR$(27)+CHR$(56)
    +CHR$(19)                                '3/72" REVERSE LINE FEED
110 FFD$=CHR$(30)+CHR$(20)+CHR$(27)+CHR$(28)+CHR$(19)        '6/72" FORWARD LF
120 LL$="TEMPERATURE RANGE (DEGREES FAHRENHEIT)"

```

```

130 XB$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(59)+STRING$(2,255) 'BOTTOM X AXIS
140 XT$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(203)+STRING$(2,255) 'TOP X AXIS
150 FL$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(188)+CHR$(156) 'FREEZING LINE
160 S1$=STRING$(33,128) '33-DOT BLANK
170 XS$=XB$+STRING$(3,136)+FL$+CHR$(27)+CHR$(16)+CHR$(1)+CHR$(200)+STRING$(3,136)
    )+XT$ 'HASHMARKS
180 PD$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(40) '40th DOT-COLUMN
190 FOR N=1 TO 2: LPRINT CHR$(30);CHR$(27);CHR$(19);CHR$(27);CHR$(32);CHR$(27);C
    HR$(16);CHR$(0);CHR$(220-4*LEN(LL$));CHR$(27);CHR$(23);CHR$(27);CHR$(14);LL$
    ;CHR$(27);CHR$(15);CHR$(27);CHR$(19);: NEXT N
200 LPRINT CHR$(13); 'LINE FEED
210 LPRINT CHR$(27);CHR$(31); 'BOLD PRINTING
220 GOSUB 4000 'SCALE NUMBERING
230 GOSUB 5000 'LEFT Y AXIS
240 LPRINT XB$;FL$;XT$;CHR$(13);
250 READ D$,L,H
260 IF D$="XXX" THEN LPRINT XB$;FL$;XT$;CHR$(13);: GOSUB 6000: GOSUB 7000: GOSUB
    8000: GOTO 990
270 L=4*L+60: H=4*H+60: D=H-L
280 GOSUB 2000 'PRINT DATE OF MONTH
290 GOSUB 1000
300 LPRINT CHR$(18);XS$;CHR$(27);CHR$(16);CHR$(QL);CHR$(LL);CHR$(255);
310 LPRINT CHR$(27);CHR$(32); 'END BOLD PRINTING
320 FOR N=1 TO (D-1)/2: LPRINT CHR$(213);CHR$(235);: NEXT N
330 LPRINT CHR$(27);CHR$(31); 'RESUME BOLD PRINTING
340 LPRINT CHR$(255);
350 LPRINT CHR$(13);
360 GOTO 250
500 '***** DATA (DATE, LOW, HIGH) *****
501 DATA NOV 1,47,51
502 DATA NOV 2,35,50
503 DATA NOV 3,28,42
504 DATA NOV 4,24,30
505 DATA NOV 5,28,33
506 DATA NOV 6,23,44
507 DATA NOV 7,31,62
508 DATA NOV 8,29,48
509 DATA NOV 9,37,40
510 DATA NOV 10,40,53
511 DATA NOV 11,39,58
512 DATA NOV 12,19,57
513 DATA NOV 13,14,24
514 DATA NOV 14,13,25
515 DATA NOV 15,12,34
516 DATA NOV 16,19,40
517 DATA NOV 17,15,47
518 DATA NOV 18,21,47
519 DATA NOV 19,47,57
520 DATA NOV 20,41,63
521 DATA NOV 21,24,47
522 DATA NOV 22,22,46
523 DATA NOV 23,12,38
524 DATA NOV 24,9,26

```

```

525 DATA NOV 25,21,39
526 DATA NOV 26,14,34
527 DATA NOV 27,11,33
528 DATA NOV 28,25,35
529 DATA NOV 29,33,36
530 DATA NOV 30,35,43
531 DATA XXX,1,2
990 LPRINT CHR$(30);CHR$(27);CHR$(32);          'END BOLD PRINTING
999 END
1000 '***** POSITIONING CONVERSIONS *****
1010 QY=Y/256: QY%=QY: YY=Y-256*QY%: QL=L/256: QL%=QL: LL=L-256*QL%
1020 IF INT(YY)=10 OR INT(YY)=12 OR INT(YY)=96 THEN YY=YY+1
1030 IF INT(LL)=10 OR INT(LL)=12 OR INT(LL)=96 THEN LL=LL+1
1040 RETURN
2000 '***** DATE-PRINTING SUBROUTINE *****
2010 IF VAL(RIGHT$(D$,2))=1 THEN LPRINT PD$;N1$;STRING$(9,128);STRING$(5,136);
2020 IF VAL(RIGHT$(D$,2))=5 THEN LPRINT PD$;N5$;STRING$(9,128);STRING$(5,136);
2030 IF VAL(RIGHT$(D$,2))=10 THEN LPRINT PD$;RLF$;CHR$(18);N1$;PD$;FFD$;CHR$(18)
;N0$;RLF$;CHR$(18);STRING$(9,128);STRING$(5,136);CHR$(30);STRING$(66,"-");C
HR$(18);
2040 IF VAL(RIGHT$(D$,2))=15 THEN LPRINT PD$;RLF$;CHR$(18);N1$;PD$;FFD$;CHR$(18)
;N5$;RLF$;CHR$(18);STRING$(9,128);STRING$(5,136);
2050 IF VAL(RIGHT$(D$,2))=20 THEN LPRINT PD$;RLF$;CHR$(18);N2$;PD$;FFD$;CHR$(18)
;N0$;RLF$;CHR$(18);STRING$(9,128);STRING$(5,136);CHR$(30);STRING$(66,"-");C
HR$(18);
2060 IF VAL(RIGHT$(D$,2))=25 THEN LPRINT PD$;RLF$;CHR$(18);N2$;PD$;FFD$;CHR$(18)
;N5$;RLF$;CHR$(18);STRING$(9,128);STRING$(5,136);
2070 IF VAL(RIGHT$(D$,2))=30 THEN LPRINT PD$;RLF$;CHR$(18);N3$;PD$;FFD$;CHR$(18)
;N0$;RLF$;CHR$(18);STRING$(9,128);STRING$(5,136);
2080 RETURN
3000 '***** "YNUMBERS" -- 5 X 7 NUMBER STRINGS *****
3010 N1$=CHR$(156)+STRING$(4,136)+CHR$(140)+CHR$(136)  "'1"
3020 N2$=CHR$(190)+CHR$(130)+CHR$(132)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
;N1$  "'2"
3030 N3$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
;N2$  "'3"
3040 N4$=STRING$(2,144)+CHR$(190)+CHR$(148)+CHR$(152)+STRING$(2,144)
;N3$  "'4"
3050 N5$=CHR$(156)+CHR$(162)+STRING$(2,160)+CHR$(158)+CHR$(130)+CHR$(190)
;N4$  "'5"
3060 N6$=CHR$(156)+CHR$(162)+CHR$(162)+CHR$(158)+CHR$(130)+CHR$(162)+CHR$(156)
;N5$  "'6"
3070 N7$=STRING$(3,132)+CHR$(136)+CHR$(144)+CHR$(160)+CHR$(190)
;N6$  "'7"
3080 N8$=CHR$(156)+STRING$(2,162)+CHR$(156)+STRING$(2,162)+CHR$(156)
;N7$  "'8"
3090 N9$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(188)+CHR$(162)+CHR$(162)+CHR$(156)
;N8$  "'9"
3100 N0$=CHR$(156)+STRING$(5,162)+CHR$(156)  "'0"
3110 NP$=CHR$(128)+STRING$(2,136)+CHR$(190)+STRING$(2,136)+CHR$(128)
;N9$  "'+"
3120 NM$=STRING$(3,128)+CHR$(156)+STRING$(3,128)  "'-"
3130 RETURN

```

```

4000 '***** NUMBERING OF LEFT Y AXIS *****
4010 GOSUB 3000
4020 LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(201);CHR$(18);N1$;CHR$(13);
4030 LPRINT CHR$(18);STRING$(97,128);N1$;S1$;N2$;S1$;N3$;S1$;N4$;S1$;N5$;S1$;N6$
      ;S1$;N7$;S1$;N8$;S1$;N9$;S1$;N0$;CHR$(13);
4040 FOR Y=57 TO 457 STEP 40: GOSUB 1000: LPRINT CHR$(27);CHR$(16);CHR$(QY);CHR$
      (YY);N0$;: NEXT Y
4050 LPRINT CHR$(13);CHR$(27);CHR$(50);CHR$(27);CHR$(50);
4060 RETURN
5000 '***** LEFT Y-AXIS *****
5010 LPRINT CHR$(18);XT$;FL$;XB$;
5020 FOR N=1 TO 10: LPRINT STRING$(19,131);CHR$(159);STRING$(19,131);CHR$(255);:
      NEXT N
5030 LPRINT CHR$(13);
5040 RETURN
6000 '***** RIGHT Y-AXIS *****
6010 LPRINT CHR$(18);XT$;FL$;XB$;STRING$(19,224);CHR$(252);STRING$(35,224);
6020 FOR N=1 TO 4: LPRINT CHR$(255);STRING$(35,224);CHR$(252);STRING$(35,224);:
      NEXT N
6030 LPRINT CHR$(255);STRING$(35,224);CHR$(252);STRING$(20,224);
6040 LPRINT CHR$(13);CHR$(27);CHR$(50);CHR$(27);CHR$(50);
6050 RETURN
7000 '***** RIGHT Y-AXIS NUMBERING *****
7010 LPRINT CHR$(18);STRING$(113,128);NM$;STRING$(137,128);NP$;STRING$(65,128);N
      P$;STRING$(65,128);NP$;CHR$(13);
7020 LPRINT STRING$(113,128);N1$;STRING$(65,128);N0$;STRING$(65,128);N1$;STRING$
      (65,128);N2$;STRING$(65,128);N3$;CHR$(13);
7030 LPRINT STRING$(113,128);N0$;STRING$(137,128);N0$;STRING$(65,128);N0$;STRIN
      G$(65,128);N0$;CHR$(13);CHR$(13);
7040 RETURN
8000 '***** RIGHT Y-AXIS LABELING *****
8010 SP$=STRING$(2,128)
8020 A$=CHR$(159)+CHR$(164)+CHR$(196)+CHR$(164)+CHR$(159)+SP$
8030 C$=CHR$(162)+STRING$(3,193)+CHR$(190)+SP$
8040 D$=CHR$(190)+STRING$(3,193)+CHR$(255)+SP$
8050 E$=STRING$(2,193)+STRING$(2,201)+CHR$(255)+SP$
8060 G$=CHR$(174)+STRING$(2,201)+CHR$(193)+CHR$(190)+SP$
8070 H$=CHR$(255)+STRING$(3,136)+CHR$(255)+SP$
8080 I$=CHR$(193)+CHR$(255)+CHR$(193)+SP$
8090 N$=CHR$(255)+CHR$(132)+CHR$(136)+CHR$(144)+CHR$(255)+SP$
8100 R$=CHR$(177)+CHR$(202)+CHR$(204)+CHR$(200)+CHR$(255)+SP$
8110 S$=CHR$(166)+STRING$(3,201)+CHR$(177)+SP$
8120 T$=STRING$(2,192)+CHR$(255)+STRING$(2,192)+SP$
8130 FOR N=1 TO 2: LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(0);CHR$(150);CHR$(27);
      CHR$(32);CHR$(27);CHR$(23);CHR$(27);CHR$(14);CHR$(18);E$;D$;A$;R$;G$;I$;T$;
      N$;E$;C$;STRING$(7,128);S$;E$;E$;R$;G$;E$;D$;CHR$(30);CHR$(27);CHR$(15);CHR
      $(27);CHR$(19);: NEXT N
8140 RETURN

```

By far the most familiar kind of floating-bar graph is the typical plotting of stock prices or stock market averages. One style of stock plotting was shown in the first illustration in

this book (figure 1-1). That figure also provides an illustration of a tandem graph with two different measures having radically different scales—daily volume, scaled on the right side of the graph, and daily stock price measure, scaled on the left.

In the WALLSTOK program (program 8-9), the DATA lines give the date; high, low, and closing prices; and sales volume. Lines 250 and 260 convert the three price values and the volume measure to dot column values that jibe with the scales on the left and right Y axes. For the plotting of each day's (or if you wish, week's) values, V% gives the number of dot columns for the height of the volume bar (printed by line 270), and H%, L%, and C% represent the high, low, and close prices. Another variable, A%, is defined as the difference between the highs and lows each day, that is, the range of prices.

A somewhat narrow solid bar, produced by `STRING$(V%,190)`, is used for plotting the volume measure. An even narrower bar (to make room for the closing-price hash mark), printed by `STRING$(A%-2,C)` in line 350, is used for the range of prices, and the bottom position of that bar is determined by L% (via the QL and LL values in line 330), the same way as in the TEMPBARS program. The printing of this high-low (range) bar is preceded by the printing of the closing-price hash mark using direct positioning (via the QC and CC values in line 280) of the five-dot code, `CHR$(252)`.

Some stock plotters like to put hash marks at the top and bottom of their high-low-closing bars, and others like the closing price to be emphasized. These features require only small changes in the programming. Still another style, used by *USA Today*, has pointed bars that are fat and have the numerical value of the closing price printed on them and the numerical values of the high and low prices above and below them. This is decidedly more difficult to program, because of the positioning of homemade perpendicular numbers in three different locations for each day, and probably not worth the effort.

The rest of the WALLSTOK program is largely housekeeping, but there's a lot of it in terms of program lines. Again, number and letter sets from homemade alphabets are needed—five-by-seven perpendicular numbers (subroutines 3000 and 5000) for the left and right Y axes, five-by-seven upside-

down letters (subroutine 7000) for the label Volume on the right axis, and seven-by-ten bold perpendicular letters (subroutine 9000) for printing the name of the corporation in the upper left corner of the graph.

PROGRAM 8-9. WALLSTOK. The standard price-and-volume stock market graph.

```

10 'PROGRAM 8-9: "WALLSTOK" -- PRICE-AND-VOLUME STOCK PLOT
15 'Copyright (c) 1985 by John Warner Davenport
18 'FOR TRS-80 COMPUTERS AND (WIDEBODY) PRINTERS
20 CLEAR 1000      '(NOT NECESSARY FOR MODEL 4)
30 PS$=CHR$(27)+CHR$(16)
40 P0$=CHR$(27)+CHR$(16)+CHR$(0)
50 P1$=CHR$(27)+CHR$(16)+CHR$(1)
60 P2$=CHR$(27)+CHR$(16)+CHR$(2)
70 CR$=P0$+CHR$(0)      'DIRECT-POSITIONED CARRIAGE RETURN
80 FOR N=1 TO 2: LPRINT CHR$(30);CHR$(27);CHR$(32);CHR$(27);CHR$(14);P1$;CHR$(60
);"PRICE/SHARE";CHR$(27);CHR$(15);: NEXT N: LPRINT CHR$(13);CHR$(13);
90 LPRINT CHR$(30);CHR$(27);CHR$(19);CHR$(27);CHR$(31);
100 GOSUB 3010      'NUMBERING OF Y AXIS
110 FOR Y=40 TO 760 STEP 40
120 GOSUB 1000 : LPRINT CHR$(18);PS$;CHR$(QY);CHR$(YY);CHR$(159);
      'HASHMARKS
130 NEXT Y
140 LPRINT P0$;CHR$(40);STRING$(250,131);STRING$(250,131);STRING$(220,131);
      'Y-AXIS
150 XL$=CHR$(18)+P0$+CHR$(39)+STRING$(2,255)      'LOWER X AXIS
160 XU$=CHR$(18)+P2$+CHR$(247)+STRING$(2,255)      'UPPER X AXIS
170 LPRINT XU$;XL$;CHR$(13);XU$;XL$;CHR$(13);
180 INPUT "NAME OF STOCK";NM$
190 GOSUB 6000
200 FOR Z=BS TO 1 STEP -1: LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);:
      NEXT Z
210 READ D$,V,H,L,C
220 IF D$="DEC 32" THEN GOSUB 4000 : GOSUB 5000 : GOSUB 7000 : GOTO 995
230 IF D$="UN" THEN GOTO 370
240 TS=80: S=640/TS: PRINT D$,V;H;L;C;S
250 C%=S*C+40: L%=S*L+40: H%=S*H+40: A%=H%-L%: V%=.8*V/50
260 IF A%<2 THEN A%=2
270 LPRINT P0$;CHR$(40);CHR$(18);STRING$(V%,190);
      ' PRINTS
      VOLUME BAR
280 QC=FIX(C%/256): CC=FIX(C%-256*QC)
290 IF CC=10 OR CC=12 OR CC=96 THEN CC=CC+1      'DETOUR
300 G=140: GG=140      'DOT-CODES FOR HIGH-LOW BAR STYLE
310 IF RIGHT$(D$,1)="2" OR RIGHT$(D$,1)="4" OR RIGHT$(D$,1)="6" OR RIGHT$(D$,1)=
"8" OR RIGHT$(D$,1)="0" THEN LPRINT XL$;CR$;CHR$(30);CHR$(27);CHR$(17);D$;CH
R$(27);CHR$(19);CHR$(18);
      ' PRINTS DATE, BOTTOM
      X AXIS
320 LPRINT CHR$(18);PS$;CHR$(QC);CHR$(CC);CHR$(252);
      ' PRINTS CLOSING PRICE

```

```

330 QL=FIX(L%/256): LL=FIX(L%-256*QL)
340 IF LL=10 OR LL=12 OR LL=96 THEN LL=LL+1      'DETOUR
350 LPRINT CHR$(18);PS$;CHR$(QL);CHR$(LL);CHR$(GG);STRING$(A%-2,G);CHR$(GG);
    'PRINTS HIGH-LOW BAR
360 GOTO 380
370 LPRINT CR$;CHR$(30);CHR$(27);CHR$(20);"UNAVAIL.";CHR$(27);CHR$(19);CHR$(18);
    P0$;CHR$(39);STRING$(2,255);
380 VT$=P0$+CHR$(36)+CHR$(18)+STRING$(3,136): HD$=CHR$(140)+STRING$(79,128)    'V
    ERTICAL TIC-MARK, DOT
390 LPRINT VT$;XL$;P0$;CHR$(120);HD$;HD$;HD$;HD$;P1$;CHR$(184);HD$;HD$;HD$;HD$;X
    U$;CHR$(13);    'X AXES, HORIZONTAL DASHES (GRID LINE)
400 IF D$="JAN 31" OR D$="FEB 28" OR D$="MAR 29" OR D$="APR 30" THEN GOSUB 2010
410 GOTO 430 : LPRINT P0$;CHR$(39);STRING$(2,143);
420 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(56);CHR$(19);
430 GOTO 210
500 ***** DATA (DATE, VOLUME, HIGH, LOW, CLOSE ) *****
505 DATA JAN 2,2176,32.875,29.125,30.250
510 DATA JAN 3,3986,31.500,28.625,28.750
515 DATA JAN 4,2254,31.125,28.750,29.000
520 DATA JAN 7,4443,30.750,27.125,27.500
525 DATA JAN 8,3500,30.000,25.000,26.000
530 DATA JAN 9,5300,35.000,30.000,32.500
535 DATA JAN 10,7500,40.000,35.000,37.500
540 DATA JAN 11,10100,43.000,37.500,41.500
545 DATA JAN 14,9100,42.500,38.000,40.500
550 DATA JAN 15,6000,39.000,35.000,37.000
555 DATA JAN 16,6500,41.500,34.000,34.500
560 DATA JAN 17,4400,43.000,37.000,39.000
565 DATA JAN 18,7500,46.000,42.000,45.500
.
.
.
900 DATA APR 23,7100,43.500,38.000,39.500
905 DATA APR 24,5200,40.000,37.500,39.625
910 DATA APR 25,4000,41.000,35.500,39.750
915 DATA APR 26,2900,39.000,34.000,37.000
920 DATA APR 29,3200,39.500,35.750,37.750
925 DATA APR 30,3800,41.125,36.000,36.500
930 DATA MAY 1,2700,39.000,35.000,35.500
935 DATA MAY 2,4200,39.500,33.500,36.125
940 DATA MAY 3,3500,40.625,35.625,37.000
945 DATA MAY 4,3400,41.500,36.875,38.125
950 DATA DEC 32,0,0,0,0
995 PRINT FRE(Z$): END
1000 QY=FIX(Y/256): YY=FIX(Y-256*QY)
1005 IF YY=10 OR YY=12 OR YY=96 THEN YY=YY+1
1010 RETURN
2000 '***** QUARTERLY VERTICAL DASHED LINE *****
2010 LPRINT CHR$(18);P0$;CHR$(41);
2020 FOR Z=1 TO 120
2030 LPRINT CHR$(128);STRING$(3,129);STRING$(2,128);
2040 NEXT Z
2050 RETURN

```

```

3000 ***** NUMBERING OF LEFT Y AXIS *****
3010 LPRINT CHR$(18);P0$;CHR$(117);CHR$(156);STRING$(3,136);CHR$(140);CHR$(136);
      "1"
3020 LPRINT P0$;CHR$(197);CHR$(190);CHR$(130);CHR$(132);CHR$(152);CHR$(160);CHR$(
      162);CHR$(156);      "2"
3030 LPRINT P1$;CHR$(21);CHR$(156);CHR$(162);CHR$(160);CHR$(152);CHR$(160);CHR$(
      162);CHR$(156);      "3"
3040 LPRINT P1$;CHR$(101);CHR$(144);CHR$(144);CHR$(190);CHR$(148);CHR$(152);CHR$(
      144);CHR$(144);      "4"
3050 LPRINT P1$;CHR$(181);CHR$(156);CHR$(162);CHR$(160);CHR$(160);CHR$(158);CHR$(
      130);CHR$(190);      "5"
3060 LPRINT P2$;CHR$(5);CHR$(156);STRING$(2,162);CHR$(158);CHR$(130);CHR$(162);C
      HR$(156);      "6"
3070 LPRINT P2$;CHR$(85);STRING$(3,132);CHR$(136);CHR$(144);CHR$(160);CHR$(190);
      "7"
3080 LPRINT P2$;CHR$(165);CHR$(156);STRING$(2,162);CHR$(156);STRING$(2,162);CHR$(
      156);      "8"
3090 LPRINT P2$;CHR$(245);CHR$(156);CHR$(162);CHR$(160);CHR$(188);STRING$(2,162)
      ;CHR$(156);CHR$(13);      "9"
3100 FOR Y=37 TO 757 STEP 80: GOSUB 1000 : LPRINT PS$;CHR$(QY);CHR$(YY);CHR$(156
      );STRING$(5,162);CHR$(156);: NEXT Y      "0"
3110 GOSUB 8030
3120 RETURN
4000 ***** RIGHT Y-AXIS *****
4010 LPRINT CHR$(18);XU$;XL$;
4020 FOR N=1 TO 18: LPRINT STRING$(39,224);CHR$(252);: NEXT N
4030 LPRINT CHR$(13);CHR$(27);CHR$(50);CHR$(27);CHR$(50);
4040 RETURN
5000 ***** "YNUMBERS" -- 5 X 7 NUMBER STRINGS *****
5010 N1$=CHR$(156)+STRING$(4,136)+CHR$(140)+CHR$(136)      "1"
5020 N2$=CHR$(190)+CHR$(130)+CHR$(132)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      "2"
5030 N3$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      "3"
5040 N4$=STRING$(2,144)+CHR$(190)+CHR$(148)+CHR$(152)+STRING$(2,144)
      "4"
5050 N5$=CHR$(156)+CHR$(162)+STRING$(2,160)+CHR$(158)+CHR$(130)+CHR$(190)
      "5"
5060 N6$=CHR$(156)+CHR$(162)+CHR$(162)+CHR$(158)+CHR$(130)+CHR$(162)+CHR$(156)
      "6"
5070 N7$=STRING$(3,132)+CHR$(136)+CHR$(144)+CHR$(160)+CHR$(190)
      "7"
5080 N8$=CHR$(156)+STRING$(2,162)+CHR$(156)+STRING$(2,162)+CHR$(156)
      "8"
5090 N9$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(188)+CHR$(162)+CHR$(162)+CHR$(156)
      "9"
5100 N0$=CHR$(156)+STRING$(5,162)+CHR$(156)      "0"
5110 MC$=STRING$(3,162)+STRING$(2,170)+CHR$(182)+CHR$(162)
5120 IC$=CHR$(156)+STRING$(5,136)+CHR$(156)
5130 LC$=CHR$(190)+STRING$(6,130)
5140 KC$=CHR$(162)+CHR$(146)+CHR$(138)+CHR$(134)+CHR$(138)+CHR$(146)+CHR$(162)
5150 LPRINT P0$;CHR$(37);CHR$(18);N0$;STRING$(33,128);N2$;STRING$(33,128);N5$;ST
      RING$(33,128);N7$;STRING$(33,128);N1$;CHR$(13);

```

```

5160 LPRINT P0$;CHR$(77);N5$;STRING$(33,128);N0$;STRING$(33,128);N5$;STRING$(33,
128);MC$;CHR$(13);
5170 LPRINT P0$;CHR$(77);N0$;STRING$(33,128);N0$;STRING$(33,128);N0$;STRING$(33,
128);IC$;CHR$(13);
5180 LPRINT P0$;CHR$(77);KC$;STRING$(33,128);KC$;STRING$(33,128);KC$;STRING$(33,
128);LC$;CHR$(13);CHR$(13);
5190 RETURN
6000 IF NM$="DANDY CORP." THEN Q=2: Y=190: BS=18: GOSUB 9040 : GOSUB 9010 : GOSUB
B 9140 : GOSUB 9040 : GOSUB 9250 : GOSUB 8030 : GOSUB 9030 : GOSUB 9150 : G
OSUB 9180 : GOSUB 9160 : GOSUB 9270
6010 RETURN
7000 REM -- UPSIDE-DOWN 5 X 7 LETTERS -- "VOLUME" LABEL --
7010 SP$=STRING$(2,128)
7020 E$=STRING$(2,193)+STRING$(2,201)+CHR$(255)+SP$
7030 L$=STRING$(4,129)+CHR$(255)+SP$
7040 M$=CHR$(255)+CHR$(160)+CHR$(152)+CHR$(160)+CHR$(255)+SP$
7050 O$=CHR$(190)+STRING$(3,193)+CHR$(190)+SP$
7060 U$=CHR$(254)+STRING$(3,129)+CHR$(254)+SP$
7070 V$=CHR$(224)+CHR$(156)+CHR$(131)+CHR$(156)+CHR$(224)+SP$
7080 FOR N=1 TO 2: LPRINT CHR$(30);P0$;CHR$(80);CHR$(27);CHR$(32);CHR$(27);CHR$(
14);CHR$(18);E$;M$;U$;L$;O$;V$;CHR$(30);CHR$(27);CHR$(15);CHR$(27);CHR$(19)
;: NEXT N
7090 RETURN
8000 '***** LINE FEEDS *****
8010 LPRINT CHR$(18);CHR$(13): FOR X=1 TO 4: LPRINT CHR$(30);CHR$(20);CHR$(27);C
HR$(30);: NEXT X: LPRINT CHR$(27);CHR$(56);: RETURN
8020 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(56);CHR$(19);: RETURN '9/72" LI
NE FEED
8030 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(28);CHR$(27);CHR$(28);CHR$(19);: RET
URN '12/72" LINE FEED
9000 '*** PRINTING OF 7 X 10 BOLD PERPENDICULAR LETTERS ***
9010 LPRINT CHR$(18);PS$;CHR$(Q);CHR$(Y);STRING$(3,227);STRING$(2,255);STRING$(2
,227);CHR$(182);STRING$(2,156);: GOSUB 8020 : RETURN ' "A"
9020 LPRINT CHR$(18);PS$;CHR$(Q);CHR$(Y);CHR$(191);CHR$(255);STRING$(2,227);STRI
NG$(2,191);STRING$(2,227);CHR$(255);CHR$(191);: GOSUB 8020 : RETURN ' "B"
9030 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(190);CHR$(255);CHR$(
227);STRING$(4,131);CHR$(227);CHR$(255);CHR$(190);: GOSUB 8020 : RETURN
' "C"
9040 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(191);CHR$(255);STRIN
G$(6,227);CHR$(255);CHR$(191);: GOSUB 8020 : RETURN ' "D"
9050 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(255);CHR$(255);CHR$(
131);CHR$(131);CHR$(159);CHR$(159);CHR$(131);CHR$(131);CHR$(255);CHR$(255);
: GOSUB 8020 : RETURN ' "E"
9060 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,131);CHR$(159);
CHR$(159);CHR$(131);CHR$(131);CHR$(255);CHR$(255);: GOSUB 8020 : RETURN
' "F"
9070 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(190);CHR$(255);CHR$(
227);CHR$(227);CHR$(251);CHR$(251);CHR$(131);CHR$(227);CHR$(255);CHR$(190);
: GOSUB 8020 : RETURN ' "G"
9080 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,227);CHR$(255);
CHR$(255);CHR$(28);CHR$(4);CHR$(227);: GOSUB 8020 : RETURN
' "H"

```

```

9090 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(188);CHR$(188);STRIN
    G$(6,152);CHR$(188);CHR$(188);: GOSUB 8020 : RETURN
    ' "I"
9100 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(158);CHR$(191);CHR$(
    179);CHR$(179);STRING$(4,176);CHR$(248);CHR$(248);: GOSUB 8020 : RETURN
    ' "J"
9110 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(227);CHR$(179);CHR$(
    155);CHR$(143);CHR$(135);CHR$(135);CHR$(143);CHR$(155);CHR$(179);CHR$(227);
    : GOSUB 8020 : RETURN
    ' "K"
9120 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(255);CHR$(255);STRIN
    G$(8,1319);: GOSUB 8020 : RETURN
    ' "L"
9130 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,227);CHR$(235);
    CHR$(235);CHR$(255);CHR$(247);CHR$(227);CHR$(227);: GOSUB 8020 : RETURN
    ' "M"
9140 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(227);CHR$(227);CHR$(
    227);CHR$(243);CHR$(251);CHR$(239);CHR$(231);CHR$(227);CHR$(227);CHR$(227);
    : GOSUB 8020 : RETURN
    ' "N"
9150 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(190);CHR$(255);STRIN
    G$(6,227);CHR$(255);CHR$(190);: GOSUB 8020 : RETURN
    ' "O"
9160 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,131);CHR$(191);
    CHR$(255);CHR$(227);CHR$(227);CHR$(255);CHR$(191);: GOSUB 8020 : RETURN
    ' "P"
9170 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(144);CHR$(190);CHR$(
    191);CHR$(251);STRING$(4,227);CHR$(255);CHR$(190);: GOSUB 8020 : RETURN
    ' "Q"
9180 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,227);CHR$(191);
    CHR$(191);CHR$(227);CHR$(227);CHR$(255);CHR$(191);: GOSUB 8020 : RETURN
    ' "R"
9190 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(190);CHR$(255);CHR$(
    227);CHR$(224);CHR$(254);CHR$(191);CHR$(131);CHR$(227);CHR$(255);CHR$(190);
    GOSUB 8020 : RETURN
    ' "S"
9200 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(8,152);CHR$(254);
    CHR$(254);: GOSUB 8020 : RETURN
    ' "T"
9210 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(190);CHR$(255);STRIN
    G$(8,227);: GOSUB 8020 : RETURN
    ' "U"
9220 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(136);CHR$(156);CHR$(
    156);CHR$(182);CHR$(182);CHR$(182);STRING$(4,227);: GOSUB 8020 : RETURN
    ' "V"
9230 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(227);CHR$(227);CHR$(
    247);CHR$(255);CHR$(235);CHR$(235);STRING$(4,227);: GOSUB 8020 : RETURN
    ' "W"
9240 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(227);CHR$(227);CHR$(
    247);CHR$(190);CHR$(156);CHR$(156);CHR$(190);CHR$(247);CHR$(227);CHR$(227);
    : GOSUB 8020 : RETURN
    ' "X"
9250 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(5,152);CHR$(188);
    CHR$(254);STRING$(3,230);: GOSUB 8020 : RETURN
    ' "Y"
9260 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(255);CHR$(255);CHR$(
    131);CHR$(134);CHR$(140);CHR$(152);CHR$(176);CHR$(224);CHR$(255);CHR$(255);
    : GOSUB 8020 : RETURN
    ' "Z"
9270 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);CHR$(140);CHR$(140);: GOS
    UB 8020 : RETURN
    ' PERIOD, POINT
9280 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(Q);CHR$(Y);STRING$(4,128);CHR$(190);
    CHR$(190);: GOSUB 8020 : RETURN
    ' HYPHEN, MINUS SIGN

```

As presented here, the WALLSTOK program is hardly a very general one. It can handle stocks with prices up to ninety dollars per share, but not those over ninety dollars, and the Y-axis scale would not work very well with stocks selling under twenty dollars, let alone penny stocks. Also, not every corporation is named Dandy Corporation. In addition, you might want to have dashed vertical grid lines (see line 400 and subroutine 2000) occurring every week or quarterly instead of monthly.

It wouldn't take very many changes to make this program much more general, though. Line 400 can easily be altered for changing the grid lines. For spelling and printing any stock's name, subroutine 6000 can be expanded to a form like the WRDCRAFT program. And at the top of the program, the user can be asked not only for the name of the stock but also for the top of the scale likely to cover all of that stock's prices. Top-of-scale (TS) values of 10, 20, 40, 80, and 160 can then determine the scale factor (S) for prices, by the formula $S = 640/TS$, and the high-low bars would be plotted in accordance with the scale thus selected. In addition, of course, new subroutines for numbering the left Y axis, from 0–10 to 0–160, would be needed.

The area graph with bars. Another kind of graph occasionally seen in business graphics is the area graph. This is usually a form of line graph, which is not very easy to program for a dot-matrix printer, but a bar-graph version of it is fairly simple.

Just to suggest the feasibility of this, figure 8-10 presents an imitation area graph produced by the AREABARS program (program 8-10). This program differs from our regular bar-graph programs in having narrow bars only four dots wide and allowing no space between the bars. The bars are "stacked" somewhat like segmented bars in horizontal bar graphs, with the bottom segment being solid black, the middle segment shaded, and the upper segment left blank. To get a better contrast between the shaded and black segments, the program prints the entire graph in bold, except when the middle segment's shading is being printed, and inserts a small white space at the top of each black segment.

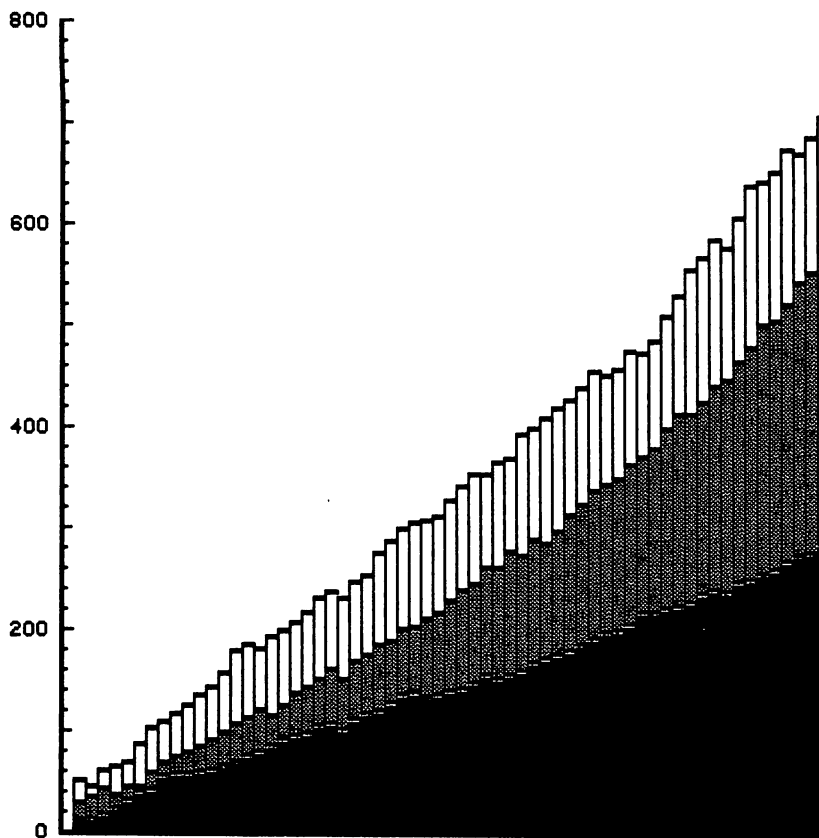


FIGURE 8-10. An area graph, in narrow-bar style (AREABARS program for the DMP-400).

PROGRAM 8-10. AREABARS. Area graph using thin segmented bars.

```

10 'PROGRAM 8-10: "AREABARS" -- AREA GRAPH USING SEGMENTED BARS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 COMPUTERS AND PRINTERS
30 CLEAR 1000
40 LPRINT CHR$(30);CHR$(27);CHR$(23)                'ELITE PITCH
50 LPRINT CHR$(27);CHR$(31);                          'BOLD PRINTING
60 LF$=CHR$(30)+CHR$(20)+CHR$(13)+CHR$(27)+CHR$(30)+CHR$(19) 'LINE FEED
70 CR$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)             'CARRIAGE RETURN
80 GOSUB 2000 : GOSUB 4000                             'Y-AXIS SUBROUTINES
100 READ A,B,C                                         'READ THREE VALUES (CUMULATIVE COMPONENTS OF A TOTAL)
110 IF A=999 THEN GOTO 280
120 A=.5*A: B=.5*B: C=.5*C                          '(SCALE FACTOR = .5)
130 M=B-A: H=C-B
140 HD=C-PC
150 PRINT PC
160 PRINT A,B,C,HD
170 LPRINT LF$;
180 IF HD<0 THEN GOSUB 1010 : LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(QC);CHR$(CC)
    );STRING$(ABS(HD),129);CR$;
190 LPRINT CHR$(18);STRING$(99,128);

```

```

200 FOR N=1 TO A-1: LPRINT CHR$(18);CHR$(19);: NEXT N: LPRINT CHR$(128);CHR$(19
1);
      'BLACK BAR
210 LPRINT CHR$(30);CHR$(8);CHR$(2);CHR$(27);CHR$(32);CHR$(18);
      'BACKSPACE, UNBOLD
220 FOR Y=0 TO M STEP 2: LPRINT CHR$(149)+CHR$(171);: NEXT Y '50%-SHADED BAR
230 LPRINT CHR$(30);CHR$(27);CHR$(31);CHR$(18); 'RETURN TO BOLD
240 LPRINT STRING$(2,191);
250 LPRINT STRING$(H,129);STRING$(2,191); 'WHITE BAR
260 PC=C ' (PC = PREVIOUS C VALUE)
270 GOTO 100
280 PRINT "JOB FINISHED": END
500 DATA 15,30,48
505 DATA 17,35,41
510 DATA 21,43,58
.
.
.
800 DATA 276,540,664
805 DATA 278,550,680
810 DATA 288,572,700
815 DATA 999,999,999
1000 '***** CONVERSIONS OF C *****
1010 QC=FIX((C+104)/256): CC=FIX((C+104)-256*QC): RETURN
2000 '***** "YNUMBERS" -- 5 X 7 NUMBER STRINGS *****
2010 N1$=CHR$(156)+STRING$(4,136)+CHR$(140)+CHR$(136) ' "1"
2020 N2$=CHR$(190)+CHR$(130)+CHR$(132)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      ' "2"
2030 N3$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      ' "3"
2040 N4$=STRING$(2,144)+CHR$(190)+CHR$(148)+CHR$(152)+STRING$(2,144)
      ' "4"
2050 N5$=CHR$(156)+CHR$(162)+STRING$(2,160)+CHR$(158)+CHR$(130)+CHR$(190)
      ' "5"
2060 N6$=CHR$(156)+CHR$(162)+CHR$(162)+CHR$(158)+CHR$(130)+CHR$(162)+CHR$(156)
      ' "6"
2070 N7$=STRING$(3,132)+CHR$(136)+CHR$(144)+CHR$(160)+CHR$(190)
      ' "7"
2080 N8$=CHR$(156)+STRING$(2,162)+CHR$(156)+STRING$(2,162)+CHR$(156)
      ' "8"
2090 N9$=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(188)+CHR$(162)+CHR$(162)+CHR$(156)
      ' "9"
2100 N0$=CHR$(156)+STRING$(5,162)+CHR$(156) ' "0"
2110 RETURN
3000 '***** INTEGRATED AXIS WITH RULER-TYPE HASHMARKS *****
3010 HA$=STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(1
43)+STRING$(9,131)+CHR$(143)+STRING$(9,131)
3020 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(99);STRING$(2,255);: FOR N=1
TO 4: LPRINT HA$;CHR$(191);HA$;CHR$(255);: NEXT N
3030 RETURN
4000 '***** 5 X 7 PERPENDICULAR NUMBERING *****
4010 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(197);CHR$(18);N2$;STRING$(93,128);N4$
;STRING$(93,128);N6$;STRING$(93,128);N8$;CHR$(13); ' "2, 4, 6, 8" (1ST COLU
MN OF NUMBERS)
4020 FOR Y=197 TO 497 STEP 100: Y1%=Y/256: Y2%=Y-256*Y1%: LPRINT CHR$(27);CHR$(1

```



```

        6);CHR$(Y1%);CHR$(Y2%);N0$;: NEXT Y  "0" (2ND COLUMN)
4030 LPRINT CHR$(13);          'LINE FEED (GRAPHIC)
4040 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(97);N0$;STRING$(93,128);N0$;STRING$(9
      3,128);N0$;STRING$(93,128);N0$;STRING$(93,128);N0$;          "0" (3RD COLUMN)
4050 LPRINT CHR$(30);CHR$(13);  'LINE FEED (NON-GRAPHIC) AND CARRIAGE RETURN
4060 GOSUB 3000                'Y AXIS WITH HASHMARKS
4070 RETURN

```

Pictograms

Remember the good old pictogram from your grade-school days? Actually, pictograms are not obsolete—you can still find them in *Time*, *USA Today*, and other slick publications these days. Whatever the vintage or usefulness of pictograms, the computer and printer do well with them.

There are at least two types of pictograms—the symbol graph and the true pictogram. The symbol graph uses repeated symbols, such as dollar signs, to present data that

PROGRAM 8-11. SYMBLGRF. Horizontal pictogram using built-in symbols.

```

10 'PROGRAM 8-11: "SYMBLGRF" -- SYMBOL GRAPH (TRS-80 MODEL 4)
20 GOSUB 500                'DEFINITION OF SYMBOLS
30 SP$=CHR$(30)+CHR$(27)+CHR$(3)  'BLANK SPACE
40 READ L$,V                'READ LABEL, VALUE
50 IF L$="END" THEN GOTO 170
60 GOSUB 600                'SYMBOL SELECTION
70 W=INT(V/10)
80 LPRINT L$;
90 LPRINT TAB(10);
100 FOR Z=1 TO W
110   LPRINT SYM$;SP$;
120 NEXT Z
130 LPRINT CHR$(30);CHR$(32);V;CHR$(10);CHR$(13);  'PRINT NUMERICAL VALUE
140 GOTO 40
150 DATA NUMBERS,256,STARS,120,MONEY,188,BOXES,210,DIAMONDS,310
160 DATA END,999
170 PRINT "JOB DONE": END
500 '***** BIT-IMAGE SYMBOL STRINGS *****
510 BX$=CHR$(18)+CHR$(255)+STRING$(5,193)+CHR$(255)+CHR$(30)
520 DM$=CHR$(18)+CHR$(136)+CHR$(148)+CHR$(162)+CHR$(193)+CHR$(162)+CHR$(148)+CHR
      $(136)+CHR$(30)
530 RETURN
600 '***** SYMBOL SELECTION *****
610 IF L$="MONEY" THEN SYM$="$"+CHR$(27)+CHR$(2): RETURN
620 IF L$="BOXES" THEN SYM$=BX$: RETURN
630 IF L$="DIAMONDS" THEN SYM$=DM$: RETURN
640 IF L$="NUMBERS" THEN SYM$="#"+CHR$(27)+CHR$(2): RETURN
650 IF L$="STARS" THEN SYM$="*"+CHR$(27)+CHR$(2): RETURN
660 RETURN

```



```

140   IF N=>0 AND N<999 THEN AB(R,C)=128+N: C=C+1: BC=BC+1: GOTO 130
150   READ M
160   FOR X=1 TO -N: AB(R,C)=128+M: C=C+1: BC=BC+1: NEXT X: GOTO 130
170   NEXT C
180   PRINT "END OF ROW";R;" BYTE COUNT=";BC;: BC=0: PRINT
190   NEXT R
200   FOR R=1 TO 4: FOR C=1 TO 60: PRINT AB(R,C);: NEXT: NEXT 'DISPLAY ARRAY
210   LPRINT CHR$(30);TAB(40);CHR$(27);CHR$(14);"PICTOGRAM";CHR$(27);CHR$(15);CHR$(13)
220   FOR I=1 TO 7
230     READ L$(I),V 'READ LABEL, VALUE
240     V1=FIX(V/10): V2=6*FIX(V-10*V1) 'CALC. # OF WHOLE FIGS., PARTIAL PORTION
250     FOR R=1 TO 4
260       IF R=2 THEN LPRINT CHR$(30);L$(I); 'PRINT LABEL ON SECOND PASS
270       LPRINT P$;
280       FOR Z=1 TO V1
290         FOR C=1 TO 60
300           LPRINT CHR$(18);CHR$(AB(R,C)); 'PRINT 1/4th OF EACH WHOLE FIGURE
310         NEXT C
320         LPRINT STRING$(3,128);
330       NEXT Z
340       LPRINT STRING$(3,128);
350       FOR C=1 TO V2
360         LPRINT CHR$(18);CHR$(AB(R,C)); 'PRINT 1/4th OF PARTIAL FIGURE
370       NEXT C
380       IF R=2 THEN LPRINT CHR$(30);CHR$(32);V; 'PRINT NUMERICAL VALUE
390       LPRINT LF$;
400     NEXT R
410     LPRINT LF$;LF$;LF$;
420   NEXT I
430   END
440 '***** DATA LINES *****
450   DATA -7,0,64,64,96,96,80,80,72,72,-2,100,-2,114,57,57,17,33,33,34,-4,66,68,-
460     4,4,-5,8,-5,16,-5,32,-5,64,-6,0,999
470   DATA 64,96,80,72,68,66,97,112,120,124,126,-7,127,-5,126,-5,124,-2,120,-3,121
480     ,-2,113,-3,114,98,98,-2,100,-2,116,92,88,72,64,32,16,80,104,72,5,101,51,19,1
490     2,112,0,999
490   DATA 127,-4,64,-5,59,-5,7,-5,15,-5,31,-5,63,-6,127,95,79,71,67,65,32,16,8,4,
500     2,1,0,1,2,4,11,87,75,40,39,16,16,8,15,999
510   DATA -5,0,-5,1,-5,2,-5,4,-5,8,-5,16,-5,32,-5,64,127,32,32,16,16,8,8,4,4,2,2,
520     1,1,-7,0,999
530   DATA EPSON,98,TANDY,90,APPLE,75,C.ITOH,49,NEC,32,OKIDATA,25,STAR,24

```

True pictograms. The PICTOGRM program (program 8-12) uses only bit-image drawings as symbols, and in the case of the pictogram displayed in figure 8-12, each drawing of the symbol takes four sweeps of the printhead, which are separated by line feeds. A two-dimensional array, identified in line 30 as AB(4,60), is used to store the picture symbol. The storage method, carried out in lines 120–190 using the data

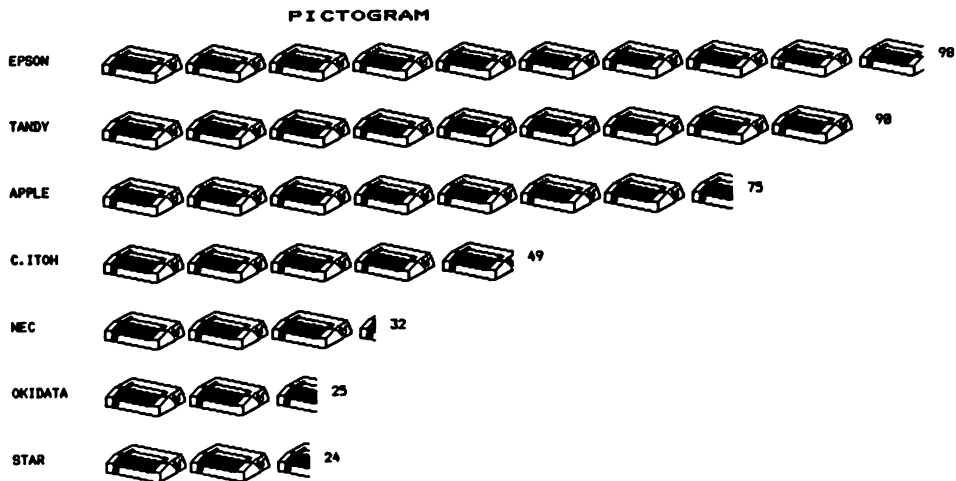


FIGURE 8-12. A pictogram with the cutoff feature for partial drawing of figures (PICTOGRM program for the DMP-400).

in lines 450–480, is much like that used in the TYP-ARRAY program of chapter 7 (program 7-3).

In the printout, the (fictitious) data in line 490 are represented by whole and partial drawings of the dot-matrix printer symbols. Again (as in the symbol graph), each drawing stands for ten units of the measure, but now the final drawing in each row is chopped off in accordance with the number of units left over after the last even multiple of ten. In line 240, V1 represents the number of whole multiples of ten, and V2 represents what is left over. The formula $V2 = 6 * \text{FIX}(V - 10 * V1)$ is used to translate the leftover portion into dot columns; for each numerical unit there are six dot columns, since each whole drawing of a printer (representing ten units) is sixty dot columns wide.

After these calculations for each of the seven data entries, lines 250–400 print the whole (lines 280–330) and fractional (lines 350–370) pictures, separated by three blank dot columns. Each entire row of whole and partial pictures is printed from the array in four printhead passes, with each pass ending at the same cutoff point of the partial picture at the right end of the picture row. The picture rows are separated by three $\frac{7}{16}$ -inch line feeds (see lines 50 and 410).

More ambitious pictograms would use a different picture for each item being compared. This would mean setting up more two-dimensional arrays for storing more sets of dot

codes and changing the print commands in lines 220–420 so that the program would shift to a new array after each row of pictures.

Program Conversions

Since most of the programs in this and the following chapters are written for TRS-80 systems, appendix C provides checklists for IBM-Epson and Apple users to use in converting the programs. In addition to changing control codes, adjusting for different print densities, and translating from one form of BASIC to another, the conversions will also involve rescaling of graphs and diagrams when programs designed for fifteen-inch printers are being adapted for eight-inch printers. Many of the conversions should be fairly easy to do if appendix C's checklists are used in conjunction with figures 4-1 and 4-2 and other materials in chapters 2–4.

Line and Range Graphs

THE defining features of line graphs are data points and their connecting lines. But connecting lines can connect other things in a graph besides the main data points—when you have one set of lines connecting the lowest values in successive sets of data values and another set of lines connecting the highest values, you have what we call a range graph. And when we combine the ingredients of a line graph with those of a range graph, we end up with a point-and-range graph.

Usually these kinds of graphs show how something changes over days, months, or years, so they're just as likely to be called time-series graphs. But there can be line or range graphs that have little to do with the time dimension and shouldn't be called time-series graphs. Now that you may be feeling more confused about the terminology, we'd better proceed to some examples.

Line Graphs

The simplest forms. If there is anything difficult about programming line graphs, it is getting those connecting lines right where they should be. So the best way to start this topic is to take the simplest form of line graph—one with just connecting lines and no data point symbols. This is what the sales graph in figure 9-1 exemplifies. The program for it, LINEGRAF (program 9-1), gives you a choice of plotting the same data four different ways; option (1), “no data points,

unequal dot spacing,” is the one that produces figure 9-1's graph. The other options are (2) “no data points, near-equal dot spacing,” (3) “data points, unequal dot spacing,” and (4) “data points, near-equal dot spacing.”

With a dot-matrix printer, the values for plotting a straight connecting line between any two points are produced by subtracting the position (height on the graph) of the second point from the position of the first point, and dividing the difference (which can be positive or negative) by the number of line feeds it takes the printer to get from the first point to the second point. So the LINEGRAF program doesn't start telling the printer to print the first connecting line until the computer has read the first two data values and been told how many line-feed steps there should be from one value to the next.

In the program, the first data value is called Z and read in line 180. The second is called A and is read in line 220. It is the difference between A and Z, called I (for interval; see line 250), that is divided by the number of repetitions of line feeds (R). The upward or downward change (C) in the vertical position of the connecting line with each line feed is determined by I/R , the A - Z interval divided by the number of line feeds; this will make C a negative value (for a downward-sloping line) if Z is greater than A (because I will be negative) and positive (upward) if A is greater than Z. The size of each line feed (V), which amounts to the distance traveled along the X axis with each line feed, is calculated in terms of the number of $\frac{1}{2}$ 216-inch line feeds making up each whole line feed, so $V = 216/R$. With options one and three (unequal dot spacing), R is 18, which makes each whole line feed $\frac{12}{216}$ -inch long (216 divided by 18 is 12). In other words, the whole vertical distance of the connecting line is chopped into eighteen equal upward or downward steps, and the whole horizontal distance between data point locations is chopped into eighteen equal-sized line feeds.

After the first connecting line has been drawn from the first data-value position to the second, A becomes Z (line 370) and a new value of A is read (line 220 again). A new connecting line for this pair of vertical positions is calculated, and so forth, through the final data value. With the TRS-80, vertical positions have to be specified by two numbers in the $27 + 16 + N1 + N2$ sequence. The constantly changing posi-

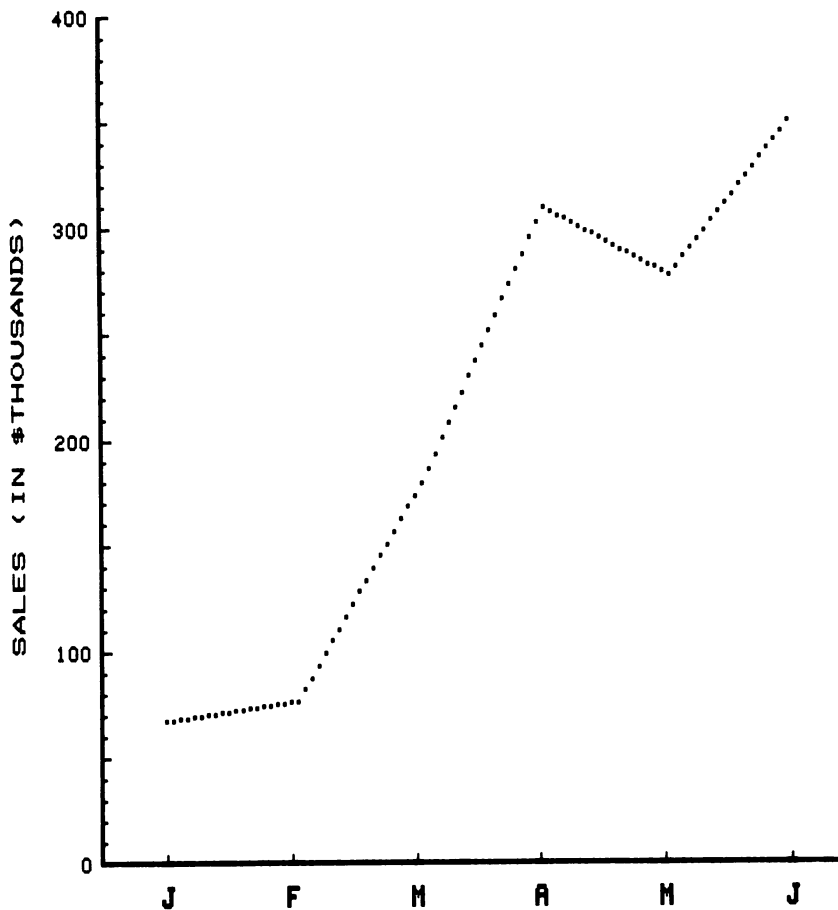


FIGURE 9-1. A simple line graph without data point symbols (LINEGRAF program, option 1).

tions of the connecting line are given by Y, which is rounded and converted in subroutine 3000 to QY% and YY% values for that positioning command. Y starts out being equal to Z (see line 280) and is incremented by C (line 330) after each print of a dot in the connecting line (line 320) and a line feed.

PROGRAM 9-1. LINEGRAF. Single-line graphs plotted four different ways.

```

10 'PROGRAM 9-1: "LINEGRAF" -- PLOTTING OF LINE GRAPHS W/ OR W/O DATAPOINTS
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR TRS-80 DMP-400 PRINTER AND MODEL III (USE BOOTHE'S DRIVER)
20 CLEAR 1000
30 P1$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(50)+STRING$(2,255)
40 PX$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(30)

```



```

50 DP=CHR$(18)+CHR$(156)+CHR$(190)+STRING$(3,255)+CHR$(190)+CHR$(156)
   ' FILLED CIRCLE (DATAPOINT SYMBOL)
60 BS=7
   ' BACKSPACE (3 1/2 DOTS)
70 LPRINT CHR$(30);CHR$(27);CHR$(23);
   ' ELITE DENSITY
80 PRINT "SELECT TYPE OF LINE GRAPH (1, 2, 3, OR 4):"
90 PRINT: PRINT "  1. NO DATAPOINTS, UNEQUAL DOT SPACING"
100 PRINT "  2. NO DATAPOINTS, NEAR-EQUAL DOT SPACING"
110 PRINT "  3. DATAPOINTS, UNEQUAL DOT SPACING"
120 PRINT "  4. DATAPOINTS, NEAR-EQUAL DOT SPACING"
130 INPUT TG
140 IF TG<1 OR TG>4 THEN PRINT "TRY AGAIN": GOTO 80
150 GOSUB 5000 : GOSUB 9000
   ' MEMORIZE NUMBERS AND LETTERS
160 GOSUB 8000
   ' Y AXIS, LABELING, AND NUMBERING
170 FOR N=1 TO 5: LPRINT P1$;CHR$(13);: NEXT N: LPRINT P1$;
   ' START OF X AXIS
180 READ XL$,Z: Z=Z+50
190 GOSUB 4000 : GOSUB 10000: LPRINT P1$;STRING$(3,136);
200 IF TG=1 OR TG=2 THEN DP=STRING$(2,140): BS=2
210 LPRINT CHR$(27);CHR$(16);CHR$(QZ%);CHR$(ZZ%);CHR$(30);CHR$(8);CHR$(BS);CHR$(
   18);DP$;
   ' PRINT FIRST DATAPOINT ONLY
220 READ XL$,A: IF A=9999 THEN GOTO 380
   ' READ X-AXIS LABEL, VALUE ON Y
230 A=A+50
240 IF TG=2 OR TG=4 THEN GOSUB 12000: C=I/R: V=216/R: GOTO 260
250 IF TG=1 OR TG=3 THEN I=A-Z: R=18: C=I/R: V=216/R
260 GOSUB 2000
270 PRINT "I=";I;" "; "V=";V;" "; "R=";R;" "; "C=";C
280 Y=Z: GOSUB 3000
290 FOR X=1 TO R
300 IF TG=1 OR TG=2 THEN CC=140: GOTO 318
310 IF X<3 OR X>(R-1) THEN CC=128 ELSE CC=140
318 FOR W=1 TO V: GOSUB 6000: NEXT W
320 LPRINT P1$;CHR$(27);CHR$(16);CHR$(QY%);CHR$(YY%);CHR$(30);CHR$(8);CHR$(2);
   CHR$(18);STRING$(2,CC);
   ' PRINT CONNECTING LINE
330 Y=Y+C: GOSUB 3000
340 NEXT X
350 IF TG=3 OR TG=4 THEN FOR W=1 TO V: GOSUB 6000 : NEXT W
360 GOSUB 10000: LPRINT P1$;STRING$(3,136);: IF TG=3 OR TG=4 THEN LPRINT CHR$(30)
   );CHR$(27);CHR$(16);CHR$(QAZ);CHR$(AAZ);CHR$(8);CHR$(BS);DP$;
   ' PRINT DATAPOINT
365 IF TG=3 THEN LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);
366 IF TG=4 THEN LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CH
   R$(27);CHR$(56);CHR$(19);
367 LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50);
370 Z=A: GOTO 220
380 FOR N=1 TO 5: LPRINT P1$;CHR$(13);: NEXT N
390 PRINT "ALL DONE": END
500 DATA J,68,F,77,M,180,A,310,M,278,J,355,J,9999
1000 '***** STRING DEFINITIONS (NUMBERS, LETTERS)
1010 RETURN
2000 '***** POSITIONING OF 1ST DATAPOINT *****
2010 AX=A: IF A>=(AX+.5) THEN AX=AX+1
   ' ROUNDING
2020 QAZ=AX/256: AAZ=AX-256*QAZ
   ' CONVERSION
2030 RETURN
3000 '***** POSITIONING OF CONNECTING LINE *****
3010 YZ=Y: IF Y>=(YZ+.5) THEN YZ=YZ+1

```

```

3020 QY%=Y%/256: YY%=Y%-256*QY%
3030 RETURN
4000 '***** POSITIONING OF DATAPOINTS *****
4010 Z%=Z: IF Z>=(Z%+.5) THEN Z%=Z%+1
4020 QZ%=Z%/256: ZZ%=Z%-256*QZ%
4030 RETURN
5000 '***** "YNUMBERS" -- 5 X 7 NUMBER STRINGS *****
5010 N1%=CHR$(156)+STRING$(4,136)+CHR$(140)+CHR$(136) ' "1"
5020 N2%=CHR$(190)+CHR$(130)+CHR$(132)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      ' "2"
5030 N3%=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(152)+CHR$(160)+CHR$(162)+CHR$(156)
      ' "3"
5040 N4%=STRING$(2,144)+CHR$(190)+CHR$(148)+CHR$(152)+STRING$(2,144)
      ' "4"
5050 N5%=CHR$(156)+CHR$(162)+STRING$(2,160)+CHR$(158)+CHR$(130)+CHR$(190)
      ' "5"
5060 N6%=CHR$(156)+CHR$(162)+CHR$(162)+CHR$(158)+CHR$(130)+CHR$(162)+CHR$(156)
      ' "6"
5070 N7%=STRING$(3,132)+CHR$(136)+CHR$(144)+CHR$(160)+CHR$(190)
      ' "7"
5080 N8%=CHR$(156)+STRING$(2,162)+CHR$(156)+STRING$(2,162)+CHR$(156)
      ' "8"
5090 N9%=CHR$(156)+CHR$(162)+CHR$(160)+CHR$(188)+CHR$(162)+CHR$(162)+CHR$(156)
      ' "9"
5100 N0%=CHR$(156)+STRING$(5,162)+CHR$(156) ' "0"
5110 RETURN
6000 LPRINT CHR$(27);CHR$(51);: RETURN ' 1/216 LINE FEED
6010 LPRINT CHR$(27);CHR$(50);: RETURN ' 1/72 LINE FEED
7000 '***** INTEGRATED AXIS WITH RULER-TYPE HASHMARKS *****
7010 HA%=STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(143)+STRING$(9,131)+CHR$(1
43)+STRING$(9,131)+CHR$(143)+STRING$(9,131)
7020 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(50);CHR$(255);: FOR N=1 TO 4
: LPRINT HA%;CHR$(191);HA%;CHR$(255);: NEXT N
7030 RETURN
8000 '***** Y AXIS WITH 5 X 7 PERPENDICULAR NUMBERING *****
8010 LPRINT CHR$(30);CHR$(27);CHR$(32); ' RESET FROM BOLD
8020 LPRINT CHR$(30);TAB(32);CHR$(27);CHR$(23);CHR$(27);CHR$(14);"SALES (IN $THO
USANDS)";CHR$(27);CHR$(15);CHR$(27);CHR$(19);CHR$(13)
8030 LPRINT CHR$(27);CHR$(31); ' BOLD PRINTING
8040 GOSUB 5000 ' GET CODE SEQUENCES OF NUMBERS
8050 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(147);CHR$(18);N1%;STRING$(93,128);N2%
;STRING$(93,128);N3%;STRING$(93,128);N4%;CHR$(13); ' "2, 4, 6, 8" (1ST COLU
MN OF NUMBERS)
8060 FOR Y=147 TO 447 STEP 100: Y1%=Y/256: Y2%=Y-256*Y1%; LPRINT CHR$(27);CHR$(1
6);CHR$(Y1%);CHR$(Y2%);N0%;: NEXT Y ' "0" (2ND COLUMN)
8070 LPRINT CHR$(13); ' LINE FEED (GRAPHIC)
8080 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(47);N0%;STRING$(93,128);N0%;STRING$(9
3,128);N0%;STRING$(93,128);N0%;STRING$(93,128);N0%; ' "0" (3RD COLUMN)
8090 LPRINT CHR$(30);CHR$(13); ' LINE FEED (NON-GRAPHIC)
8100 GOSUB 7000 ' Y AXIS WITH HASHMARKS
8110 RETURN
9000 '***** 7 X 10 BOLD PERPENDICULAR LETTERS *****
9010 A%=PX%+STRING$(4,227)+CHR$(255)+CHR$(255)+CHR$(227)+CHR$(227)+CHR$(190)+CHR
$(190)

```

```

9020 D$=PX$+CHR$(191)+CHR$(255)+CHR$(28)+CHR$(6)+CHR$(227)+CHR$(255)+CHR$(191)
9030 F$=PX$+CHR$(28)+CHR$(4)+CHR$(131)+CHR$(159)+CHR$(159)+CHR$(131)+CHR$(131)+C
    HR$(255)+CHR$(255)
9040 J$=PX$+CHR$(158)+CHR$(191)+STRING$(2,179)+STRING$(4,176)+STRING$(2,248)
9050 M$=PX$+STRING$(4,227)+CHR$(235)+CHR$(235)+CHR$(255)+CHR$(247)+STRING$(2,227
    )
9060 N$=PX$+STRING$(3,227)+CHR$(243)+CHR$(251)+CHR$(239)+CHR$(231)+STRING$(3,227
    )
9070 O$=PX$+CHR$(190)+CHR$(255)+STRING$(6,227)+CHR$(255)+CHR$(190)
9080 P$=PX$+STRING$(4,131)+CHR$(191)+CHR$(255)+STRING$(2,227)+CHR$(255)+CHR$(191
    )
9090 S$=PX$+CHR$(190)+CHR$(255)+CHR$(227)+CHR$(224)+CHR$(254)+CHR$(191)+CHR$(131
    )+CHR$(227)+CHR$(255)+CHR$(190)
9100 RETURN
10000 '***** X-AXIS LETTERING *****
10010 GOSUB 9000
10020 IF XL$="A" THEN LPRINT PX$;A$;: RETURN
10030 IF XL$="D" THEN LPRINT PX$;C$;: RETURN
10040 IF XL$="F" THEN LPRINT PX$;F$;: RETURN
10050 IF XL$="J" THEN LPRINT PX$;J$;: RETURN
10060 IF XL$="M" THEN LPRINT PX$;M$;: RETURN
10070 IF XL$="N" THEN LPRINT PX$;N$;: RETURN
10080 IF XL$="O" THEN LPRINT PX$;O$;: RETURN
10090 IF XL$="S" THEN LPRINT PX$;S$;: RETURN
10100 RETURN
12000 '***** CALCULATIONS FOR NEAR-EQUAL DOT SPACING *****
12010 I=A-Z
12020 H=SQR(100(2+I(2))/100: PRINT "H=";H
12030 R=18*H
12040 RETURN

```

With options one and two, no data-point symbols (circles, triangles, etc.) are plotted, but the connecting lines change their direction precisely where they should have been if you had wanted them. In fact, these turning points show precisely where the *center* of each data-point symbol should be.

If you're not happy with the appearance of the sales curve in Figure 9-1, then maybe you have good taste. The connecting line is almost a solid line when it is nearly horizontal, but with steep rises it has widely spaced dots. Is there anything we can do to remedy this?

For a single-line graph, anyway, there is, or at least we can make the dots more nearly equal in their spacing. We can select options two and four, "near-equal spacing." This shifts the program to calculations of R, and hence C and V as well, so that the number of dots between data values will vary with the vertical distance that they have to traverse. With 216 $\frac{1}{2}$ 16-

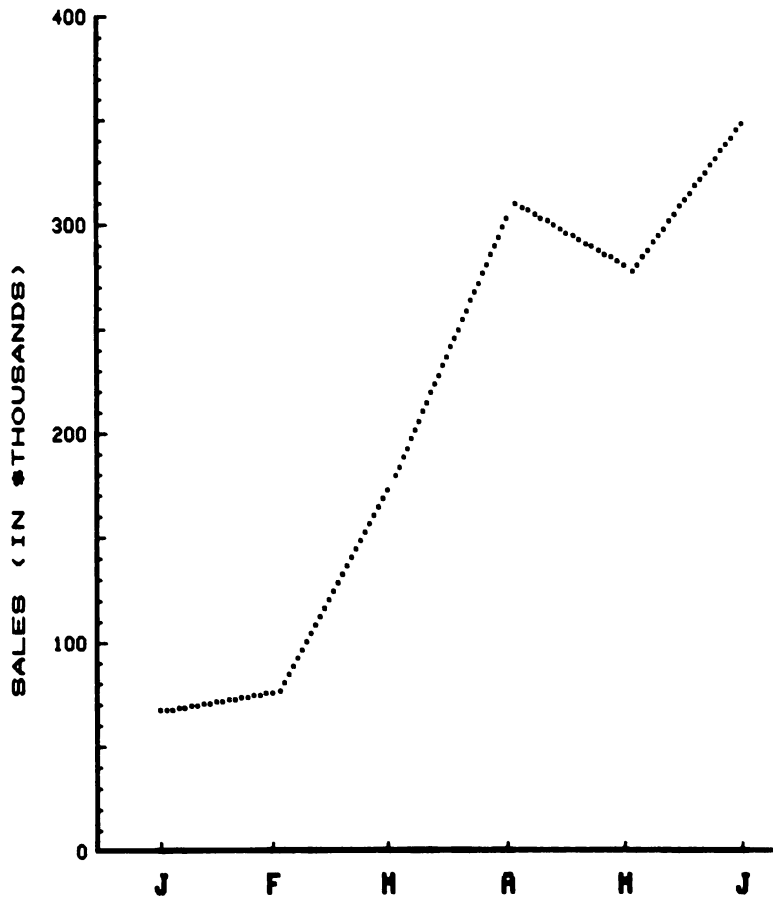


FIGURE 9-2. More nearly equal spacing of the dots in connecting lines (LINEGRAF program, option 2).

inch line feeds to play with in each inch between data values, and hundreds of dot columns for vertical positioning as well, we can calculate those combinations of line feeds and vertical change with each line feed that give us more equally spaced dots.

Option two (near-equal dot spacing without data points) was used in the plotting of the same data in figure 9-2, and option four (near-equal dot spacing with data-point symbols) was used for plotting figure 9-3. If you count the number of dots between turning points in figure 9-2, you'll find that there are quite a few more of them than in figure 9-1 where the vertical intervals are large (e.g., between the second and third or third and fourth data values). Particularly when data

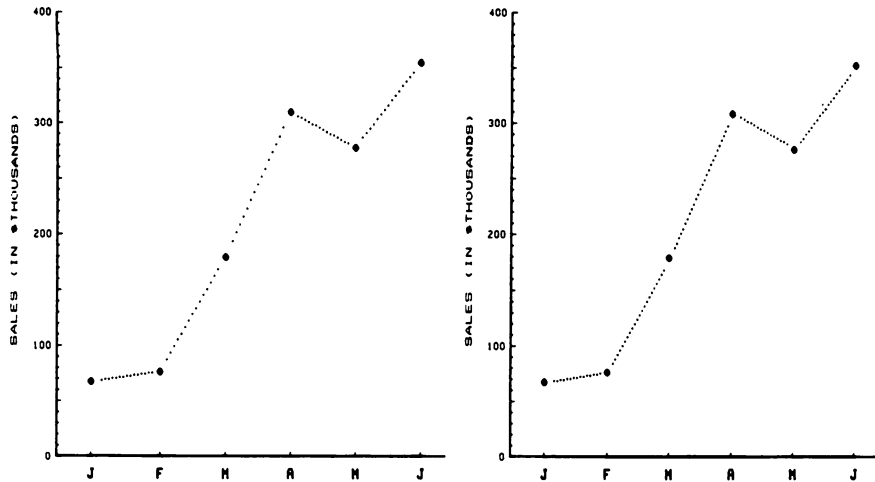


FIGURE 9-3. A line graph having data point symbols with unequal (left) and nearly equal (right) spacing of connecting-line dots (LINEGRAF program, options 3 and 4).

points make the turning points clearer, a better-looking graph results from the more-equal spacing.

The calculations for options two and four (line 240 and subroutine 12000) make use of the Pythagorean Theorem ($A^2 + B^2 = H^2$, or the square of the hypotenuse, H, in a right triangle equals the sum of the squares of the other two sides, A and B). Line 12020 solves the Pythagorean equation for H (the hypotenuse), using the values 100 and I (the A – Z interval) for the two sides. The H value, which has a minimum of

FIGURE 9-4. An assortment of data point symbols and their TRS-80 dot-code sequences.

SYMBOLS FOR DATA POINTS

- ▣ `STRING$(2,255)+STRING$(3,227)+STRING$(2,255)`
- ◆ `CHR$(136)+CHR$(156)+CHR$(190)+CHR$(255)+CHR$(190)+CHR$(156)+CHR$(136)`
- ✕ `CHR$(227)+CHR$(247)+CHR$(190)+CHR$(156)+CHR$(190)+CHR$(247)+CHR$(227)`
- ⊙ `CHR$(156)+CHR$(190)+STRING$(3,227)+CHR$(190)+CHR$(156)`
- `STRING$(7,255)`
- ◇ `CHR$(136)+CHR$(156)+CHR$(182)+CHR$(227)+CHR$(182)+CHR$(156)+CHR$(136)`

1.0, is multiplied by 18, the minimum number of line feeds allowed, to determine R, the number of dot-printing steps in the connecting line. The size of each line feed (V), which now varies as the program moves from one connecting line to the next, is determined by dividing 216 by R (line 250), as before. The overall spacing of the dots in the graph can be changed by changing the minimum number of line feeds allowed from 18 to some other number.

Data points. The line graph in figure 9-3 is a more conventional one, with data-point symbols and connecting lines. The range of symbols available is very large if you realize how easy it is to make them out of dot patterns. Figure 9-4 shows just a few of the many possible examples. Each of those shown involves a sequence of seven dot codes, but of course data-point symbols of much different size can also be used.

Even after you have mastered the programming of connecting lines, the line graph with data points is not as easy as it looks. It is all too likely that backspacing for exact positioning of the data points will be forgotten, and if that happens, the connecting lines won't be pointing directly at the centers of the data-point symbols. So notice in line 60 of the LINEGRAF program the backspace variable, BS, set to 7. This, used after CHR\$(8) in line 360, drops the circular data point $3\frac{1}{2}$ dot columns (recall that it takes two backspace units to make a difference of one dot column). Also notice the smaller backspace, CHR\$(8);CHR\$(2), used to lower the two-dot connecting line by one dot column (line 320). These backspace adjustments will make the center of each data point, not its bottom, appear at the correct height on the Y axis, and also make the connecting lines point directly at the centers, not the bottoms, of the data points. The latter can't be guaranteed by these, because extra line feeds have an annoying way of fouling up the alignment of data points with connecting lines, too.

Notice also that line graphs with data points will have a neater appearance, especially in comparison to most line graphs produced by pen plotters, if the connecting lines stop short of touching the data point. Putting in the desirable gap around each data-point print is accomplished rather easily by changing from the usual dot code for a dot in a connecting

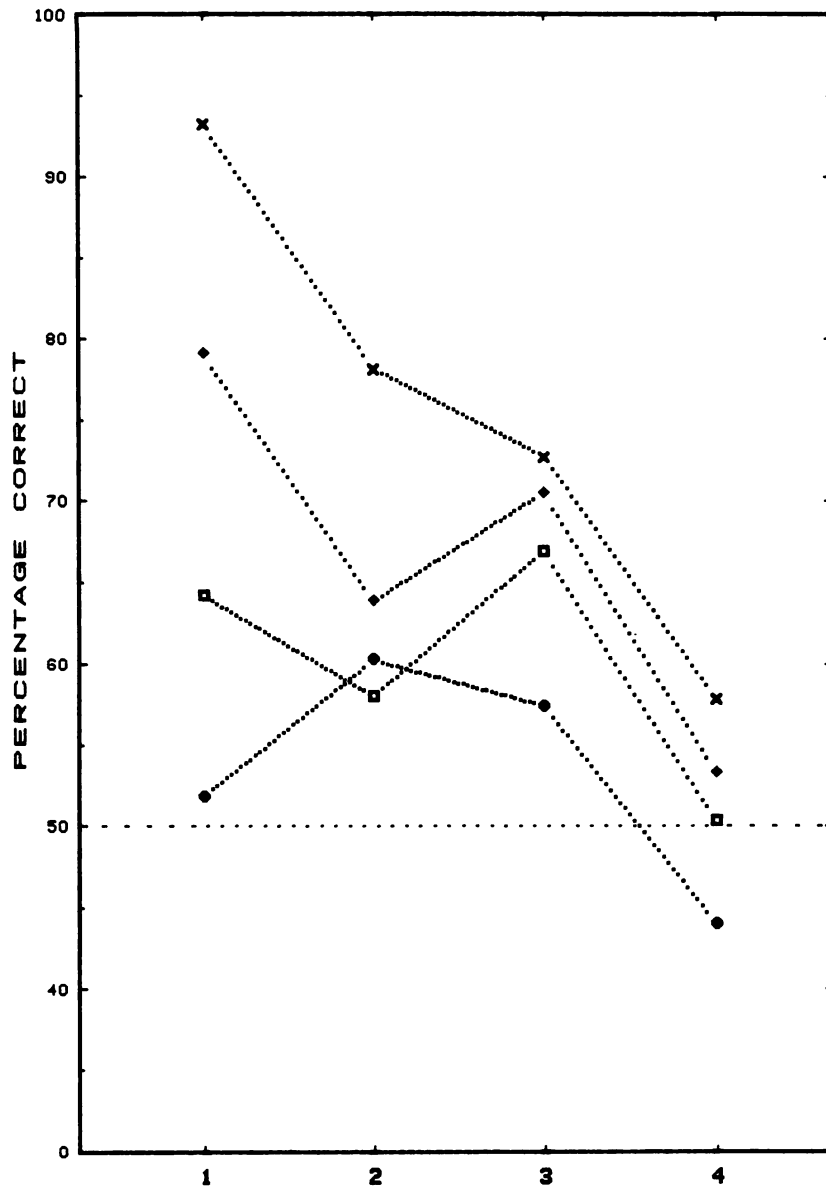


FIGURE 9-5. A multiple-line graph produced by the MULTLING program for the DMP-400.

line to a blank code (0 for IBM- and Apple-type printers, 128 for TRS-80-type printers)—see line 310.

While we're still dealing with single-line graphs, we should note another type very often seen in scientific journals. This is the kind that has many data points that are not joined by connecting lines at all but are summarized by a

continuous curve. The curve usually portrays some mathematical function, which has been arrived at through curve-fitting procedures (beyond the scope of this book). Once you have the equation for the curve, it can be plotted the same way that several curves at a time were plotted in the FUNCJAZZ programs (programs 5-3 and 5-4), before or after the data-point symbols have been plotted as in this chapter.

Multiple-line graphs. Line graphs having two or more "curves" are exemplified in figure 9-5. This is another common type of graph in the scientific journals, and like the mean-and-standard-error bar graphs of chapter 9, standard-error verticals may be added to the data points. The new challenges to be faced in going beyond single-line graphs may be seen in program 9-2, MULTLING.

PROGRAM 9-2. MULTLING. Multiple-line graph.

```

10 'PROGRAM 9-2: "MULTLING" -- MULTIPLE-LINE-GRAPH PROGRAM
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 DMP-400 PRINTER AND MODEL III (USE BOOTHE'S PRINTER DRIVER)
30 PN$=CHR$(18)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(80)      'Both DOT-COLUMN
40 BS$=CHR$(30)+CHR$(8)+CHR$(3)+CHR$(18)                  'BACKSPACE (1 1/2 DOTS)
50 ED$=CHR$(30)+CHR$(27)+CHR$(23)+CHR$(18)                  'ELITE DENSITY
60 YL$="PERCENTAGE CORRECT": TB=70-.5*LEN(YL$)
70 FOR N=1 TO 1: LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(0);CHR$(0);TAB(TB);CHR$(
    27);CHR$(14);CHR$(27);CHR$(18);YL$;CHR$(27);CHR$(15);CHR$(27);CHR$(19);: NEX
    T N
80 LPRINT CHR$(13);
90 LPRINT CHR$(30);CHR$(27);CHR$(23);: UL=800: GOSUB 2020
100 GOSUB 6010 : GOSUB 7010 : GOSUB 5100      'Y-AXIS, NUMBERING, HASHMARKS
110 GOSUB 490      'X-AXIS EXTENSION
120 CTR=0
130 READ XL$,AR,BR,CR,DR      'READ X-AXIS LABEL, FOUR RAW-SCORE VALUES
140 A=10*AR-200: B=10*BR-200: C=10*CR-200: D=10*DR-200
150 IF XL$="END" THEN GOSUB 5010
160 IF CTR>0 THEN GOTO 200
170 GOSUB 3010      'PRINT DATAPPOINTS
180 PA=A: PB=B: PC=C: PD=D
190 CTR=CTR+1: IF CTR=1 THEN GOTO 140
200 IA=A-PA: IB=B-PB: IC=C-PC: ID=D-PD: KA=IA/72: KB=IB/72: KC=IC/72: KD=ID/72
210 W=PA: X=PB: Y=PC: Z=PD
220 PRINT "PA=";PA;" "; "PB=";PB;" "; "PC=";PC;" "; "PD=";PD
230 PRINT " A="; A;" "; " B="; B;" "; " C="; C;" "; " D="; D
240 PRINT "IA=";IA;" "; "IB=";IB;" "; "IC=";IC;" "; "ID=";ID
250 PRINT "KA=";KA;" "; "KB=";KB;" "; "KC=";KC;" "; "KD=";KD
260 FOR N=2 TO 72 STEP 2
270 GOSUB 610      'PRINT CONNECTING LINES
280 NEXT N

```



```

290 GOTO 310
300 FOR N=70 TO 72 STEP 2: FOR NN=1 TO 2: GOSUB 4050 : NEXT NN: NEXT N
310 W=0: X=0: Y=0: Z=0
320 GOSUB 3010                                'PRINT DATAPOINTS
330 PA=A: PB=B: PC=C: PD=D
340 CTR=CTR+1
350 GOTO 140
360 PRINT "JOB DONE": END
370 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(99);STRING$(2,156);CHR$(27);CHR$(16);C
    HR$(QU);CHR$(UU);STRING$(2,156);: RETURN
380 DATA 1,51.94,64.38,79.22,93.33
390 DATA 2,60.40,58.16,64.00,78.25
400 DATA 3,57.58,67.08,70.65,72.88
410 DATA 4,44.12,50.40,53.44,57.92
420 DATA END,-99,-99,-99,-99
430 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(99);CHR$(255);CHR$(255);
                                'BOTTOM X-AXIS
440 GOSUB 2020 : LPRINT CHR$(27);CHR$(16);CHR$(QU);CHR$(UU);CHR$(255);CHR$(255)
    ;: GOSUB 4020
450 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(99);STRING$(2,255);
460 LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(44);CHR$(156);                                '50% MARK
470 GOSUB 2020 : LPRINT CHR$(27);CHR$(16);CHR$(QU);CHR$(UU);CHR$(255);CHR$(255)
    ;                                'TOP X-AXIS
480 RETURN
490 FOR J=1 TO 5: GOSUB 430: GOSUB 4020: NEXT J
500 RETURN
510 FOR J=1 TO 12: GOSUB 4030: NEXT J
520 RETURN
530 QA=FIX(A/256): AA=FIX(A-256*QA)
540 RETURN
550 QB=FIX(B/256): BB=FIX(B-256*QB)
560 RETURN
570 QC=FIX(C/256): CC=FIX(C-256*QC)
580 RETURN
590 QD=FIX(D/256): DD=FIX(D-256*QD)
600 RETURN
610 '***** PRINTING OF CONNECTING LINES *****
620 IF CTR=0 AND N>70 THEN CC=128 ELSE CC=140
630 IF CTR>0 AND N<6 OR N>70 THEN CC=128 ELSE CC=140
640 QW=FIX(W/256): WW=FIX(W-256*QW)
650 LPRINT CD$;CHR$(27);CHR$(16);CHR$(QW);CHR$(WW);BS$;STRING$(2,CC);
660 W=W+2*KA
670 QX=FIX(X/256): XX=FIX(X-256*QX)
680 LPRINT CD$;CHR$(27);CHR$(16);CHR$(QX);CHR$(XX);BS$;STRING$(2,CC);
1000 X=X+2*KB
1010 QY=FIX(Y/256): YY=FIX(Y-256*QY)
1020 LPRINT CD$;CHR$(27);CHR$(16);CHR$(QY);CHR$(YY);BS$;STRING$(2,CC);
1030 Y=Y+2*KC
1040 QZ=FIX(Z/256): ZZ=FIX(Z-256*QZ)
1050 LPRINT CD$;CHR$(27);CHR$(16);CHR$(QZ);CHR$(ZZ);BS$;CHR$(128);STRING$(2,CC);
1060 Z=Z+2*KD
1070 GOSUB 370
1080 IF N=4 OR N=10 OR N=16 OR N=22 OR N=28 OR N=34 OR N=40 OR N=46 OR N=52 OR N

```

```

=58 OR N=64 OR N=70 THEN LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(44);CHR$(18)
;CHR$(140); 'GRID LINE AT 50% LEVEL
1090 GOSUB 4050 :GOSUB 4050 :GOSUB 4050
2000 RETURN
2010 '***** CONVERSIONS OF UL AND H *****
2020 QU=FIX(UL/256): UU=FIX(UL-256*QU)
2030 RETURN
2040 QH=FIX(H/256): HH=FIX(H-256*QH)
2050 RETURN
3000 '***** PRINTING OF DATAPOINTS *****
3010 GOSUB 530 : LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(QA);CHR$(AA);CHR$(8);CH
R$(7);CHR$(18);CHR$(156);CHR$(190);CHR$(255);CHR$(255);CHR$(255);CHR$(190);
CHR$(156); 'FILLED CIRCLE
3020 GOSUB 550 : LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(QB);CHR$(BB);CHR$(8);CH
R$(7);CHR$(18);CHR$(255);CHR$(255);CHR$(227);CHR$(227);CHR$(227);CHR$(255);
CHR$(255); 'OPEN SQUARE
3030 GOSUB 570 : LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(QC);CHR$(CC);CHR$(8);CH
R$(7);CHR$(18);CHR$(136);CHR$(156);CHR$(190);CHR$(255);CHR$(190);CHR$(156);
CHR$(136); 'FILLED DIAMOND
3040 GOSUB 590 : LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(QD);CHR$(DD);CHR$(8);CH
R$(7);CHR$(18);CHR$(227);CHR$(247);CHR$(190);CHR$(156);CHR$(190);CHR$(247);
CHR$(227); 'FAT "X"
3050 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(QU);CHR$(UU-3);CHR$(28);CHR$(3);CHR$
(136);CHR$(255);CHR$(255); 'TOP X-AXIS HASHMARK
3060 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(99);CHR$(255);CHR$(255);CHR$(28);CHR$
(3);CHR$(136); 'BOTTOM X-AXIS HASHMARK
3070 ON VAL(XL$) GOSUB 8010,8030,8040,8050,8060,8070,8080,8090,8100,8120
3080 RETURN
4000 '***** LINE FEEDS *****
4010 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(28);CHR$(19);: RETURN
'HALF-LINE FORWARD LINE FEED
4020 LPRINT CHR$(18);CHR$(10);: RETURN 'GRAPHIC LINE FEED
4030 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);: RETURN
'HALF-LINE REVERSE FEED
4040 LPRINT CHR$(27);CHR$(51);: RETURN '1/216 LINE FEED
4050 LPRINT CHR$(27);CHR$(50);: RETURN '1/72 LINE FEED
5000 '***** TERMINAL (RIGHT-HAND) Y AXIS *****
5010 INPUT "TERMINAL Y-AXIS (Y OR N)";YN$
5020 IF YN$="Y" THEN INPUT "NUMBER OF SPACES FOR LABELING (0, 2, 4, 6, ETC.)";NS
: GOTO 5040
5030 IF YN$="N" OR YN$="NO" THEN GOTO 360
5040 FOR X=1 TO NS/2: GOSUB 430 : GOSUB 4020 : NEXT X
5050 GOSUB 7020
5060 FOR H=200 TO UL-100 STEP 100: GOSUB 2040 : LPRINT CHR$(27);CHR$(16);CHR$(Q
H);CHR$(HH);CHR$(18);CHR$(143);: NEXT H 'LONG HASHMARKS
5070 FOR H=150 TO UL-50 STEP 100: GOSUB 2040 : LPRINT CHR$(27);CHR$(16);CHR$(QH
);CHR$(HH);CHR$(18);CHR$(140);: NEXT H 'SHORT HASHMARKS
5080 GOSUB 2020 : LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(99);CHR$(191);
CHR$(191);CHR$(27);CHR$(16);CHR$(QU);CHR$(UU);CHR$(191);CHR$(191)
'END OF X-AXES (BOTTOM & TOP)
5090 GOTO 360
5100 FOR H=200 TO UL-100 STEP 100: GOSUB 2040 : LPRINT CHR$(27);CHR$(16);CHR$(
QH);CHR$(HH);CHR$(18);CHR$(248);: NEXT H 'LONG HASHMARKS

```

```

5110 FOR H=150 TO UL-50 STEP 100: GOSUB 2040 : LPRINT CHR$(27);CHR$(16);CHR$(QH
);CHR$(HH);CHR$(18);CHR$(152);: NEXT H          'SHORT HASHMARKS
5120 FOR H=99 TO UL STEP UL-99: GOSUB 2040 : LPRINT CHR$(27);CHR$(16);CHR$(QH);
CHR$(HH);CHR$(18);CHR$(254);CHR$(254);: NEXT H    'BEGINNING OF X-AXES (BO
TTOM & TOP)
5130 GOSUB 4020
5140 RETURN
6000 '***** Y-AXIS NUMBERS (5 X 7 PERPENDICULARS *****
6010 P$=CHR$(18)+CHR$(27)+CHR$(16)
6020 LPRINT P$;CHR$(3);CHR$(29);CHR$(184);STRING$(4,144);CHR$(152);CHR$(144);
          '1"
6030 GOSUB 4020
6040 LPRINT P$;CHR$(0);CHR$(197);STRING$(2,144);CHR$(190);CHR$(148);CHR$(152);ST
RING$(2,144);
          '4"
6050 LPRINT P$;CHR$(1);CHR$(41);CHR$(156);CHR$(162);STRING$(2,160);CHR$(158);CHR
$(130);CHR$(190);
          '5"
6060 LPRINT P$;CHR$(1);CHR$(141);CHR$(156);STRING$(2,162);CHR$(154);CHR$(130);CH
R$(162);CHR$(156);
          '6"
6070 LPRINT P$;CHR$(1);CHR$(241);STRING$(3,132);CHR$(136);CHR$(144);CHR$(160);CH
R$(190);
          '7"
6080 LPRINT P$;CHR$(2);CHR$(85);CHR$(156);STRING$(2,162);CHR$(156);STRING$(2,162
);CHR$(156);
          '8"
6090 LPRINT P$;CHR$(2);CHR$(185);CHR$(156);CHR$(162);CHR$(160);CHR$(172);STRING$
(2,162);CHR$(156);
          '9"
6100 LPRINT P$;CHR$(3);CHR$(29);CHR$(156);STRING$(5,162);CHR$(156);
          '"0"
6110 GOSUB 4020
6120 FOR X=97 TO 797 STEP 100: QX=X/256: QX%=QX: XX=X-256*QX%: LPRINT P$;CHR$(Q
X%);CHR$(XX);CHR$(156);STRING$(5,162);CHR$(156);: NEXT X
6130 GOSUB 4010 :GOSUB 4010
6140 RETURN
7000 '***** AXIS LINES *****
7010 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(99);CHR$(28);CHR$(201);CHR$(
134);CHR$(28);CHR$(250);CHR$(134);CHR$(28);CHR$(250);CHR$(134);: RETURN
          'Y AXIS (LEFT)
7020 LPRINT CHR$(18);CHR$(27);CHR$(16);CHR$(0);CHR$(99);STRING$(201,176);STRING$
(250,176);STRING$(250,176);: RETURN          'Y AXIS (RIGHT)
8000 '***** 7 X 10 BOLD PERPENDICULAR NUMBERS *****
8010 LPRINT PN$;CHR$(188);CHR$(188);CHR$(28);CHR$(5);CHR$(152);CHR$(156);CHR$(15
2);CHR$(144);: RETURN          '"1"
8020 LPRINT PN$;CHR$(28);CHR$(3);CHR$(156);: RETURN          '"-"
8030 LPRINT PN$;CHR$(255);CHR$(255);CHR$(131);CHR$(134);CHR$(188);CHR$(248);CHR$
(224);CHR$(227);CHR$(255);CHR$(190);: RETURN          '"2"
8040 LPRINT PN$;CHR$(190);CHR$(255);CHR$(227);CHR$(224);CHR$(188);CHR$(188);CHR$
(224);CHR$(227);CHR$(255);CHR$(190);: RETURN          '"3"
8050 LPRINT PN$;CHR$(176);CHR$(176);CHR$(176);CHR$(255);CHR$(255);CHR$(179);CHR$
(182);CHR$(188);CHR$(184);CHR$(176);: RETURN          '"4"
8060 LPRINT PN$;CHR$(190);CHR$(255);CHR$(227);STRING$(2,224);CHR$(191);CHR$(159)
;CHR$(131);STRING$(2,255);: RETURN          '"5"
8070 LPRINT PN$;CHR$(190);CHR$(255);STRING$(2,227);CHR$(255);CHR$(191);CHR$(131)
;CHR$(227);CHR$(255);CHR$(190);: RETURN          '"6"
8080 LPRINT PN$;STRING$(3,134);CHR$(140);CHR$(152);CHR$(176);STRING$(2,224);STRI
NG$(2,255);: RETURN          '"7"
8090 LPRINT PN$;CHR$(190);CHR$(255);STRING$(2,227);STRING$(2,190);STRING$(2,227)
;CHR$(255);CHR$(190);: RETURN          '"8"

```

```

8100 LPRINT PN$;CHR$(190);CHR$(255);CHR$(227);CHR$(224);CHR$(254);CHR$(255);STRIN
    NG$(2,227);CHR$(255);CHR$(190);: RETURN    '"9"
8110 LPRINT CHR$(18);PN$;CHR$(190);CHR$(255);STRING$(6,227);CHR$(255);CHR$(190);
    : RETURN
8120 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);CHR$(27);CHR$(50);CHR$(
    18);PN$;STRING$(2,188);STRING$(5,152);CHR$(156);CHR$(152);CHR$(144);CHR$(10
    );PN$;CHR$(190);CHR$(255);STRING$(6,227);CHR$(255);CHR$(190);: RETURN
    '"10"
8130 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);: GOSUB 4050 : GOSUB 80
    30 : GOSUB 4020 : GOSUB 4050 : GOSUB 4050 : GOSUB 8110 : GOSUB 4030 : RETUR
    N    '"20"
8140 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);: GOSUB 4050 : GOSUB 80
    50 : GOSUB 4020 : GOSUB 4050 : GOSUB 4050 : GOSUB 8110 : GOSUB 4030 : RETUR
    N    '"40"

```

For one thing, using nearly equal dot spacing for connecting lines of varying slope (see figure 9-2) is out of the question when you have two or more sets of connecting lines to worry about. For another thing, you now have more than one dot print, and perhaps four or five, for drawing the connecting lines in each pass of the printhead.

But the latter turns out to be an easier programming task than you might think. With a single line feed (say $\frac{1}{12}$ -inch) applying to several curves in the graph, you simply have several differences, instead of only a single difference between two data point positions, to be divided by the number of line feeds between the data points. As with MULTLING's lines 200 and 210, you have to type in a few more letters and numbers to calculate the connecting-line dot positions, but it's hardly any more work for the computer and printer.

The higher the printing density, the more the connecting lines in line graphs will look like real lines that are closer to professional publication standards for graphs. But they will still fall far short of what you can achieve manually with plastic drafting tapes for the connecting lines. The graph in figure 9-6, which would pass muster with most scientific journals, has solid and dashed plastic tapes covering the original printer-drawn connecting lines. This graph shows that standard-error verticals can be programmed in line graphs as readily as in bar graphs. It also shows another way the computer can be used for labeling a graph: Seven-by-ten bold perpendicular letters were printed by the WRDCRAFT program (program 7-2) for the axis labels and the upper title, and pasted on the graph (in both perpendicular and upright orientations). This labeling could have been programmed (as

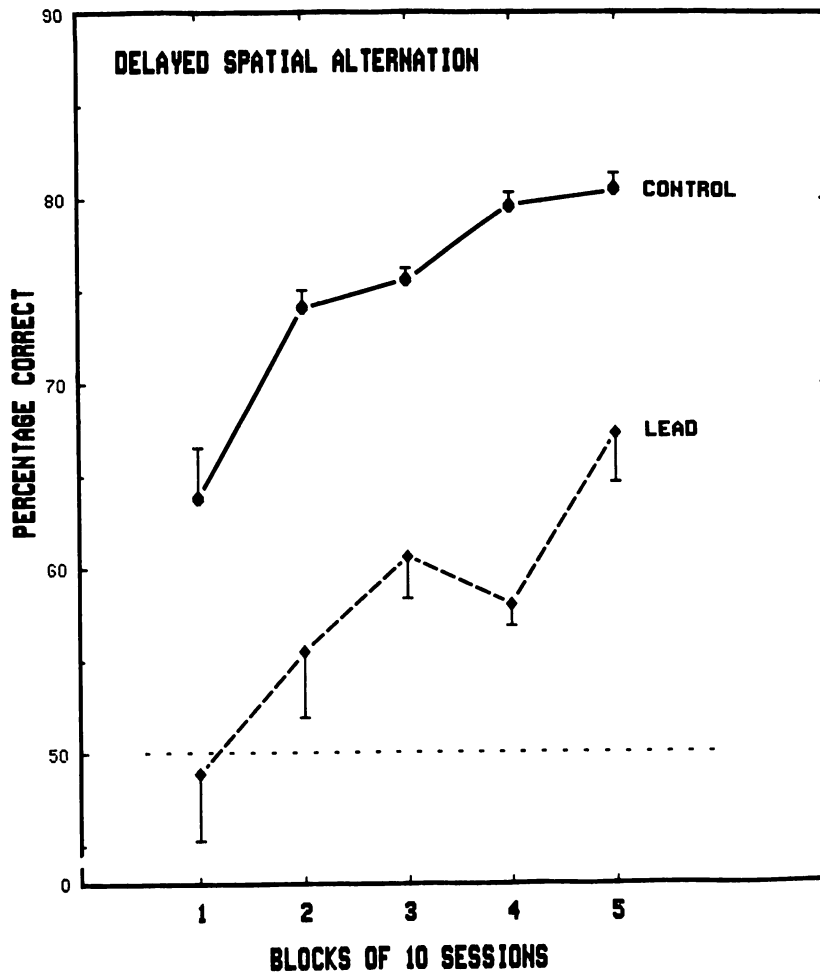


FIGURE 9-6. A MULTLING-produced graph with connecting lines made by hand with plastic drafting tapes and manually pasted labels printed by the WRDCRAFT program.

with figures 1-3 and 8-8), but sometimes under time pressure it makes more sense to do it by using the printer as a typesetter and pasting than by adding more subroutines to a program.

Alphabetic characters as data point symbols. Although the older graphics manuals say you should never have more than three or four curves in a line graph, there are times when there is no choice but to have many more. This is the case with plotting major-league baseball pennant races, where the number of different curves in the graph depends on the number of teams in the league or division. See figure 9-7, which

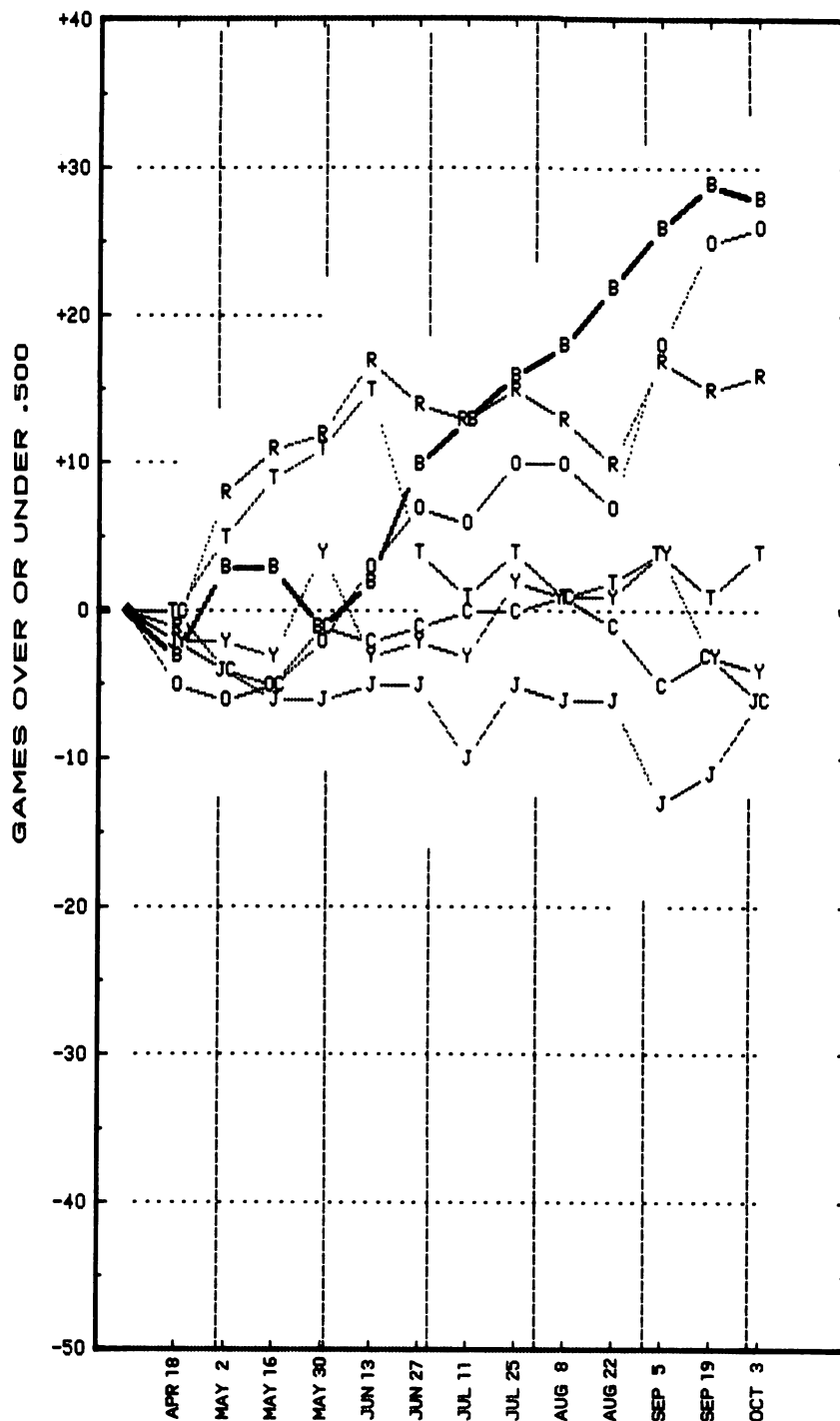


FIGURE 9-7. A baseball title-race graph having programmed connecting lines and letter abbreviations as data point symbols (the 1972 American League East race, won by the Milwaukee Brewers).

charts the progress of the seven teams in the American League Eastern Division that were battling it out for the 1982 title.

Switching from the usual data-point symbols to letters of the alphabet, which can also play the important role of abbreviations, makes the graph reader's task fairly easy, even with a seven-curve graph. The program that generated this graph uses perpendicular five-by-eleven condensed letters for the data points. It also introduces a technique for dealing with ties between teams by printing their letter symbols side by side instead of one on top of the other. This involves moving one letter a half-line-feed backward and the letter tied with it a half-feed forward. If there is a three-way tie, then whole reverse and forward line feeds place two letters on either side of a normally positioned letter.

If you want a more detailed comparison of computer-drawn vs. manually placed connecting lines and other features in a line graph, compare figure 9-7 with the pennant-race graph back in the first chapter (figure 1-4). While the data for these two figures were not the same (1982 vs. 1984 races), the programs for them were highly similar. The main differences between them are that the connecting lines in figure 1-4 were done with a variety of Formatt tapes, the grid lines (as well as the connecting lines) in figure 9-7 were done by the computer and printer, and most of the lettering and labeling in figure 1-4 was cut and pasted. This is one of several demonstrations of the fact that the more complex graphs often can't be *completely* done in a satisfactory way by the average dot-matrix printers of today—some of the finishing touches have to be done by hand for a good result—but a large part of the most detailed and tedious work in the plotting can be handled quite nicely by the printer.

Range Graphs

Figure 9-8 has our first example of a range graph. In the program for this graph (PNTRANGE, program 9-3), successive sets of scores read from a disk datafile are sorted so that lowest and highest values in each set can be identified. Connecting lines, calculated the same way as for data points, connect all the highest values and all the lowest values separately. For smoothness of the connecting lines, $\frac{1}{72}$ -inch line

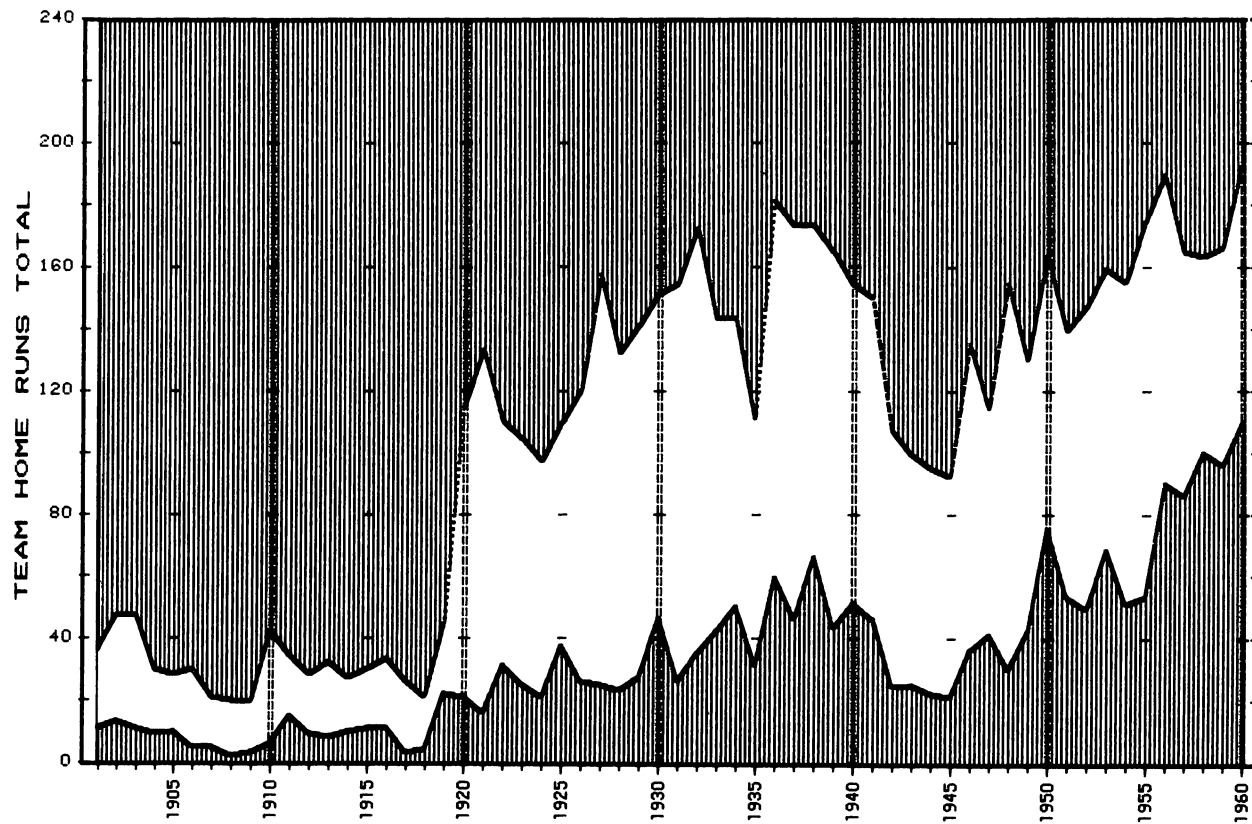


FIGURE 9-8. A range graph from the PNTRANGE program, showing the total range of team home-run totals in the American League from 1901 to 1960.

feeds are used along with condensed dot packing. To emphasize the open range area between the upper and lower limits, vertical lines extend upward from the bottom X axis and downward from the upper X axis to the range boundaries.

Such vertical lines can also be drawn with every line feed, so that a solid black area surrounds the range area, or only with every pair of range values, to make it easier to see where those values fall on the X axis. Still another option is to use shaded bars with a standard graphic line feed instead of these vertical lines, and not bother with connecting lines at all (see figure 9-11). Printing time, and wear and tear on the printer, would be greatly reduced with the latter variation, and indeed large graphs may call for programmed pauses to prevent the printer from overheating. This is especially a danger with the DMP-400 when bold, condensed printing is done contin-

PROGRAM 9-3. PNTRANGE. Point-and-range or rank-and-range graph.

```

10 'PROGRAM 9-3: "PNTRANGE" -- FOR RANGE, POINT-&-RANGE, AND RANK-&-RANGE GRAPHS
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR MICROSOFT-BASIC COMPUTERS AND DMP-400 PRINTER
40 LPRINT CHR$(30);CHR$(27);CHR$(32);CHR$(27);CHR$(19); 'RESET
50 CLEAR 2000 'NOT NECESSARY FOR COMPUTERS WITH DYNAMIC STRING STORAGE
60 P$=CHR$(27)+CHR$(16): P0$=CHR$(27)+CHR$(16)+CHR$(0) 'POSITIONING STRINGS
65 P1$=CHR$(27)+CHR$(16)+CHR$(1) ' " "
80 GOSUB 7010: GOTO 90 'GET NUMBER DEFINITIONS
82 LPRINT CHR$(18);P0$;CHR$(31);STRING$(2,254);P1$;CHR$(207);STRING$(2,254);: RE
TURN 'SEGMENT OF X AXIS
85 Y1=FIX(Y/256): Y2=FIX(Y-256*Y1): RETURN
87 FOR LF=1 TO 21: LPRINT CHR$(27);CHR$(51);: NEXT LF: RETURN '21/216" LF
88 FOR LF=1 TO 15: LPRINT CHR$(27);CHR$(51);: NEXT LF: RETURN '15/216" LF
90 INPUT "FILE NAME";FL$: GOSUB 8000 'DETERMINE # OF TEAMS (NT)
100 PRINT "SELECT STATISTIC TO BE PLOTTED:"
110 PRINT " 1. WON-LOST PERCENTAGE"
120 PRINT " 2. TEAM HOME RUNS TOTAL"
130 PRINT " 3. TEAM BATTING AVERAGE"
140 PRINT " 4. TEAM SLUGGING AVERAGE"
150 PRINT " 5. TEAM STOLEN BASES TOTAL"
160 PRINT " 6. TEAM FIELDING AVERAGE"
170 PRINT " 7. TEAM TOTAL WALKS YIELDED"
180 PRINT " 8. TEAM STRIKEOUTS PITCHED"
190 PRINT " 9. TEAM EARNED RUN AVERAGE"
200 INPUT "YOUR SELECTION (1 - 9)";V
210 ON V GOSUB 2010,2020,2030,2040,2050,2060,2070,2080,2090
220 INPUT "HIGHLIGHTED TEAM (BOS,NYY,PHI,BAL,WAS,CLE,DET,CHI,MIL,STL,KCA,KCR,TEX
,CAL,OAK, OR SEA)";T$ 'CHANGE ABBREVIATIONS FOR NATIONAL LEAGUE

```

```

230 INPUT "Y-AXIS LABELING (Y OR N)";YN$
235 INPUT "CONNECTING LINES (Y OR N)";CQ$
237 INPUT "VERTICAL LINES (Y OR N)";LQ$
238 INPUT "INTERMEDIATE VERTICAL LINES (Y OR N)";IQ$
240 IF LEFT$(YN$,1)="N" THEN GOTO 380
250 IF LEFT$(YN$,1)="Y" THEN GOTO 280
260 PRINT "STATISTIC: ";STAT$,"HIGHLIGHTED TEAM: ";T$
280 LPRINT CHR$(30);TAB(TB);CHR$(27);CHR$(14);STAT$;CHR$(27);CHR$(15);: GOSUB 87
290 ON V GOSUB 5000,5090,5190,5290,5410,5500,5600,5670,5770 'Y-AXIS LABELING
300 LPRINT CHR$(18);CHR$(10); 'LINE FEED
310 LPRINT CHR$(18);P0$;CHR$(32);STRING$(223,134);STRING$(210,134); 'Y AXIS
320 FOR Y=32 TO 470 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CHR$(Y2);CHR$
(252);: NEXT Y 'HASH MARKS
330 FOR N=1 TO 2
340 GOSUB 82
350 GOSUB 6030 'LINE FEED
360 NEXT N
370 LPRINT P0$;CHR$(32);STRING$(223,132);STRING$(210,132); 'BEGINNING VERTICAL
380 DIM YR$(NT),WP$(NT),HR$(NT),BA$(NT),SA$(NT),SB$(NT),FA$(NT),BB$(NT),SO$(NT),
ERA$(NT),C$(NT),IV$(NT+1)
390 OPEN "I", 1, FL$
400 FOR D=1 TO NT
410 INPUT #1, YR$(D),WP$(D),HR$(D),BA$(D),SA$(D),SB$(D),FA$(D),BB$(D),SO$(D),E
RA$(D)
420 PRINT YR$(D);" ";WP$(D);" ";HR$(D);" ";BA$(D);" ";SA$(D);" ";SB$(D);" ";FA
$(D);" ";BB$(D);" ";SO$(D);" ";ERA$(D)
430 NEXT D
440 ON V GOSUB 3010,3030,3050,3070,3090,3110,3130,3150,3170 'SET UP STATISTIC T
O BE SORTED
450 GOSUB 4000 'SORTING OF TEAMS ON SELECTED STATISTIC
460 LX=M*B+I: HX=M*A+I: UX=M*P+I '* DOT-COLUMNS FOR LOW, HIGH, TEAM POSITIONS
470 L1%=FIX(LX/256): L2%=FIX(LX-256*L1%)
480 M1%=FIX((LX-32)/256): M2%=FIX((LX-32)-256*M1%): IF M1%>0 THEN M1%=255 ELSE M
1%=0
490 IF UX>255 THEN Q=1 ELSE Q=0
500 IF Q=1 THEN UX=UX-256
505 IF UX=9 THEN UX=8 'DETOUR AROUND TROUBLE CODE 9
510 H1%=FIX(HX/256): H2%=FIX(HX-256*H1%)
520 UV=465-HY: UX=FIX(UV/256): UW=FIX(UV-256*UX): IF UX>0 THEN V1=255 ELSE V1=0
530 IF YR$(1)="1901" OR YR$(1)="1961" OR YR$(1)="1969" OR YR$(1)="1977" THEN GOT
O 550 ELSE GOSUB 700 'CHANGE "1961" TO "1962" FOR NATIONAL LEAGUE
540 REM -- FOR RANGE GRAPH ONLY, CHANGE "GOTO 550" TO "GOTO 580" IN LINE 530
550 ON S GOSUB 1010,1020,1030,1040,1050,1060,1070,1080,1090,1100,1110,1120,1130,
1140 'PRINT NOS. 1 TO 14 (RANK-&-RANGE GRAPH)
570 LPRINT P$;CHR$(L1%);CHR$(L2%);CHR$(30);CHR$(8);CHR$(3);CHR$(18);CHR$(136);CH
R$(156);CHR$(136); 'LOW POINT SYMBOL
575 IF LQ$="N" OR LQ$="NO" THEN GOTO 600
580 LPRINT CHR$(18);P0$;CHR$(28);STRING$(3,136);STRING$(2,190);STRING$(M1%,136);
STRING$(M2%,136); 'LOW POINT VERTICAL LINE
600 LPRINT P$;CHR$(H1%);CHR$(H2%);CHR$(30);CHR$(8);CHR$(3);CHR$(18);CHR$(136);CH
R$(156);CHR$(136); 'HIGH POINT SYMBOL
605 IF LQ$="N" OR LQ$="NO" THEN GOTO 620
610 LPRINT P$;CHR$(H1%);CHR$(H2%);STRING$(V1,136);STRING$(UW,136);CHR$(27);CHR$(
16);CHR$(1);CHR$(207);STRING$(2,190); 'HIGH POINT VERT. LINE

```

```

620 IF RIGHT$(YR$(1),1)="5" OR RIGHT$(YR$(1),1)="0" THEN LPRINT P0$;CHR$(0);CHR$(30);CHR$(27);CHR$(17);YR$(1);CHR$(27);CHR$(19);CHR$(18);P0$;CHR$(26);STRING$(5,136);
630 IF RIGHT$(YR$(1),1)="5" THEN FOR Y=102 TO 390 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CHR$(Y2);STRING$(2,128);CHR$(190);: NEXT Y
640 IF RIGHT$(YR$(1),1)="0" THEN LPRINT CHR$(30);P0$;CHR$(33);STRING$(72,"-");: FOR Y=102 TO 390 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);CHR$(18);STRING$(2,128);CHR$(255);: NEXT Y
650 IF V=1 THEN FL=156 ELSE VL=128
660 LPRINT P0$;CHR$(32);CHR$(18);CHR$(156);P0$;CHR$(248);CHR$(FL);P1$;CHR$(208);CHR$(156);
670 PL%=L$: PU%=U$: PH%=H%
680 IF EOF(1) THEN GOTO 945
690 GOTO 400
700 '***** SUBROUTINE FOR PRINTING CONNECTING LINES *****
705 IF CQ$="N" OR CQ$="NO" THEN GOSUB 9000: RETURN 'OMIT CONNECTING LINES
710 ST=32 'NUMBER OF STEPS
720 LF$=CHR$(27)+CHR$(51) '1/216" LINE FEED
730 IL=(L%-PL%)/ST: IU=(U%-PU%)/ST: IH=(H%-PH%)/ST
740 CL=PL%+IL: CU=PU%+IU: CH=PH%+IH
750 FOR CF=1 TO ST
760 C1%=FIX(CL/256): C2%=FIX(CL-256*C1%): IF C2%=9 THEN C2%=8
770 C3%=FIX(CU/256): C4%=FIX(CU-256*C3%): IF C4%=9 THEN C4%=8
780 C5%=FIX(CH/256): C6%=FIX(CH-256*C5%): IF C6%=9 THEN C6%=8
790 PRINT "CF=";CF;" CL=";CL;" CU=";CU;" CH=";CH
800 LPRINT LF$;
810 LPRINT P$;CHR$(C1%);CHR$(C2%);CHR$(30);CHR$(8);CHR$(2);CHR$(18);STRING$(2,140);P$;CHR$(C5%);CHR$(C6%);CHR$(30);CHR$(8);CHR$(2);CHR$(18);STRING$(2,140);
820 LPRINT P$;CHR$(C3%);CHR$(C4%);CHR$(30);CHR$(8);CHR$(1);CHR$(18);CHR$(136);
830 IF CF=INT(.25*ST) OR CF=INT(ST/2) OR CF=INT(.75*ST) THEN GOSUB 880
840 CL=CL+IL: CU=CU+IU: CH=CH+IH
850 NEXT CF
860 LPRINT LF$;
870 RETURN
880 '***** SUBROUTINE FOR INTERMEDIATE VERTICAL LINES *****
885 IF IQ$="N" OR IQ$="NO" THEN RETURN
890 A1%=FIX((CL-32)/256): A2%=FIX((CL-32)-256*A1%): IF A1%>0 THEN A1%=255
900 UD=465-CH: D1%=FIX(UD/256): D2%=FIX(UD-256*D1%): IF D1%=0 THEN E1%=0 ELSE E1%=255
910 LPRINT CHR$(18);P$;CHR$(0);CHR$(31);STRING$(2,190);STRING$(A1%,136);STRING$(A2%,136);
920 LPRINT P$;CHR$(C5%);CHR$(C6%);STRING$(E1%,136);STRING$(D2%,136);P$;CHR$(1);CHR$(207);STRING$(2,190);
930 RETURN
940 '***** CLOSING OF FILE, PRINTING OF TERMINAL (RIGHT) AXIS *****
945 CLOSE 1
950 PRINT "FILE CLOSED"
955 INPUT "TERMINAL Y-AXIS (Y OR N)";TY$
960 IF LEFT$(TY$,1)="N" THEN GOTO 985
965 IF LEFT$(TY$,1)="Y" THEN LPRINT CHR$(18);P0$;CHR$(31);STRING$(2,255);STRING$(222,136);STRING$(210,136);P$;CHR$(1);CHR$(207);STRING$(2,136);CHR$(10);
970 LPRINT CHR$(18);P0$;CHR$(32);STRING$(223,176);STRING$(210,176);
975 FOR Y=6 TO 78 STEP 6: LPRINT CHR$(30);TAB(Y);CHR$(245);: NEXT Y
HASHMARKS

```

```

980 LPRINT P0$;CHR$(31);CHR$(18);STRING$(2,191);P$;CHR$(1);CHR$(207);STRING$(2,1
91);
985 PRINT "JOB COMPLETED"
990 END
1000 '***** PRINTING OF NUMBERS 1-14 OR BLACK SQUARE *****
1010 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);N1$;: RE
TURN
1020 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N2$;: RETURN
1030 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N3$;: RETURN
1040 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N4$;: RETURN
1050 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N5$;: RETURN
1060 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N6$;: RETURN
1070 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N7$;: RETURN
1080 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N8$;: RETURN
1090 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;N9$;: RETURN
1100 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;NN$;: RETURN
1110 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;NE$;: RETURN
1120 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;NW$;: RETURN
1130 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(7);CHR$(18)
;NR$;: RETURN
1140 LPRINT CHR$(30);CHR$(27);CHR$(16);CHR$(Q);CHR$(U%);CHR$(8);CHR$(9);CHR$(18)
;CHR$(255);CHR$(221);CHR$(221);CHR$(197);CHR$(213);CHR$(205);CHR$(221);CHR$
(221);CHR$(255);: RETURN
2000 '***** NAME OF STATISTIC, TAB STOP FOR LABELING Y AXIS *****
2010 STAT$="WON-LOST PERCENTAGE": TB=23: RETURN
2020 STAT$="TEAM HOME RUNS TOTAL": TB=22: RETURN
2030 STAT$="TEAM BATTING AVERAGE": TB=22: RETURN
2040 STAT$="TEAM SLUGGING AVERAGE": TB=22: RETURN
2050 STAT$="TEAM STOLEN BASES TOTAL": TB=20: RETURN
2060 STAT$="TEAM FIELDING AVERAGE": TB=22: RETURN
2070 STAT$="TEAM TOTAL WALKS YIELDED": TB=19: RETURN
2080 STAT$="TEAM STRIKEOUTS PITCHED": TB=20: RETURN
2090 STAT$="TEAM EARNED RUN AVERAGE": TB=20: RETURN
3000 '***** SORTING PARAMETERS *****
3010 FOR D=1 TO NT: IV$(D)=WP$(D): NEXT D: M=.72: I=-112: RETURN
3030 FOR D=1 TO NT: IV$(D)=HR$(D): NEXT D: M=1.8: I=32: RETURN
3050 FOR D=1 TO NT: IV$(D)=BA$(D): NEXT D: M=3.6: I=-688: RETURN
3070 FOR D=1 TO NT: IV$(D)=SA$(D): NEXT D: M=1.8: I=-436: RETURN
3090 FOR D=1 TO NT: IV$(D)=SB$(D): NEXT D: M=.9: I=32: RETURN
3110 FOR D=1 TO NT: IV$(D)=FA$(D): NEXT D: M=7.2: I=-6628: RETURN
3130 FOR D=1 TO NT: IV$(D)=BB$(D): NEXT D: M=.72: I=-112: RETURN
3150 FOR D=1 TO NT: IV$(D)=SO$(D): NEXT D: M=.36: I=-40: RETURN
3170 FOR D=1 TO NT: IV$(D)=ERA$(D): NEXT D: M=72: I=-40: RETURN

```

```

4000 '***** SORTING OF TEAMS BY STATISTIC *****
4005 F=4: IF V=8 OR V=9 THEN F=5
4010 FOR D=1 TO NT: IF RIGHT$(IV$(D),3)=T$ THEN IV$(D)=LEFT$(IV$(D),F-1)+T$: GOT
    O 4030
4020 NEXT D
4030 PRINT "NOW SORTING TEAMS BY ";STAT$
4031 SWITCH=0
4032 FOR D=1 TO NT
4033   IF IV$(D)<IV$(D+1) THEN SWAP IV$(D),IV$(D+1): SWITCH=SWITCH+1
4035 NEXT D
4036 PASS=PASS+1: PRINT "PASS=";PASS
4038 IF SWITCH>0 THEN GOTO 4031 ELSE PRINT "SORTING FINISHED": PASS=0
4040 FOR Z=NT TO 1 STEP -1
4050   IF V=7 OR V=9 THEN E=(NT+1)-Z ELSE E=Z
4055   PRINT IV$(Z),Z,E
4060   C$(Z)=IV$(Z)+STR$(E)
4070   IF MID$(C$(Z),F,3)=T$ THEN P=VAL(LEFT$(C$(Z),F-1)): S=VAL(RIGHT$(C$(Z),2)
    ): PRINT T$,"P=";P,"S=";S
4080 NEXT Z
4090 A=VAL(IV$(1)): B=VAL(IV$(NT))
4110 RETURN
5000 '***** Y-AXIS LABELING *****
5010 '----- WON-LOST PERCENTAGE -----
5020 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CHR$(Y2);STR
    ING$(2,224);: NEXT Y: GOSUB 87 'DECIMAL POINTS
5040 LPRINT P0$;CHR$(30);N2$;P0$;CHR$(102);N3$;P0$;CHR$(174);N4$;P0$;CHR$(246);N
    5$;P1$;CHR$(62);N6$;P1$;CHR$(134);N7$;P1$;CHR$(206);N8$;: GOSUB 87
5060 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5065 GOSUB 87
5070 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5075 GOSUB 6030 '4/72" LINE FEED
5080 RETURN
5090 '----- TEAM HOME RUNS -----
5100 GOSUB 6030: LPRINT CHR$(18);P0$;CHR$(246);N1$;P1$;CHR$(62);N1$;
5110 FOR Y=390 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N2$;: NEXT
    Y: GOSUB 87
5130 FOR Y=102 TO 462 STEP 360: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N4$;: NEXT
    Y
5140 LPRINT P0$;CHR$(174);N8$;P1$;CHR$(62);N6$;P1$;CHR$(134);N0$;P0$;CHR$(246);N
    2$;: GOSUB 87
5160 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
    : GOSUB 6030
5180 RETURN
5190 '----- TEAM BATTING AVERAGE -----
5200 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);CHR$(18);STR
    ING$(2,224);: NEXT Y 'DECIMAL POINTS
5210 GOSUB 87: FOR Y=30 TO 318 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N2
    $;: NEXT Y
5220 FOR Y=390 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N3$;: NEXT
    Y: GOSUB 87
5230 FOR Y=30 TO 390 STEP 360: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT
    Y

```

```

5240 FOR Y=102 TO 462 STEP 360: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N2$;: NEXT
      Y      "2"
5250 LPRINT P0$;CHR$(174);N4$;P0$;CHR$(246);N6$;P1$;CHR$(62);N8$;: GOSUB 87
5260 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5270 GOSUB 6030      '4/72" LINE FEED
5280 RETURN
5290 '----- TEAM SLUGGING AVERAGE -----
5300 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CHR$(Y2);STR
      ING$(2,224);: NEXT Y: GOSUB 87      'DECIMAL POINTS
5320 LPRINT P0$;CHR$(30);N2$;P0$;CHR$(102);N3$;P0$;CHR$(174);N3$;P0$;CHR$(246);N
      3$;P1$;CHR$(62);N4$;P1$;CHR$(134);N4$;P1$;CHR$(206);N5$;: GOSUB 87
5360 FOR Y=30 TO 390 STEP 360: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N6$;: NEXT
      Y
5370 FOR Y=102 TO 462 STEP 360: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT
      Y
5380 LPRINT P0$;CHR$(174);N4$;P0$;CHR$(246);N8$;P1$;CHR$(62);N2$;: GOSUB 87
5390 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5395 GOSUB 6030      '4/72" LINE FEED
5400 RETURN
5410 '----- TEAM STOLEN BASES TOTAL -----
5420 GOSUB 6030: LPRINT CHR$(18);P0$;CHR$(174);N1$;P0$;CHR$(246);N2$;P1$;CHR$(62
      );N3$;P1$;CHR$(134);N4$;P1$;CHR$(206);N4$;
5450 GOSUB 87: LPRINT P0$;CHR$(102);N8$;P0$;CHR$(174);N6$;P0$;CHR$(246);N4$;P1$;
      CHR$(62);N2$;P1$;CHR$(134);N0$;P1$;CHR$(206);N8$;: GOSUB 87
5470 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5480 GOSUB 6030      '4/72" LINE FEED
5490 RETURN
5500 '----- TEAM FIELDING AVERAGE -----
5510 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CHR$(Y2);STR
      ING$(2,224);: NEXT Y: GOSUB 87      'DECIMAL POINTS
5530 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N9$;: NEXT Y
      : GOSUB 87
5550 LPRINT P0$;CHR$(30);N2$;P0$;CHR$(102);N3$;P0$;CHR$(174);N4$;P0$;CHR$(246);N
      5$;P1$;CHR$(62);N6$;P1$;CHR$(134);N7$;P1$;CHR$(206);N8$;: GOSUB 87
5570 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N5$;: NEXT Y
5580 GOSUB 6030      '4/72" LINE FEED
5590 RETURN
5600 '----- TEAM TOTAL WALKS YIELDED -----
5610 GOSUB 6030: LPRINT CHR$(18);P0$;CHR$(30);N2$;P0$;CHR$(102);N3$;P0$;CHR$(174
      );N4$;P0$;CHR$(246);N5$;P1$;CHR$(62);N6$;P1$;CHR$(134);N7$;P1$;CHR$(206);N8
      $;
5620 GOSUB 87: FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0
      $;: NEXT Y: GOSUB 87
5640 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5650 GOSUB 6030      '4/72" LINE FEED
5660 RETURN
5670 '----- TEAM STRIKEOUTS PITCHED -----
5680 GOSUB 87: FOR Y=318 TO 462 STEP 72: LPRINT CHR$(18);P1$;CHR$(Y-256);N1$;: N
      EXT Y: GOSUB 87
5700 LPRINT P0$;CHR$(30);N2$;P0$;CHR$(102);N4$;P0$;CHR$(174);N6$;P0$;CHR$(246);N
      8$;P1$;CHR$(62);N0$;P1$;CHR$(134);N2$;P1$;CHR$(206);N4$;: GOSUB 87
5730 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y

```

```

5740 GOSUB 87
5750 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
      'SECOND "0" COLUMN
5755 GOSUB 6030      '4/72" LINE FEED
5760 RETURN
5770 '----- TEAM EARNED RUN AVERAGE -----
5775 GOSUB 87: LPRINT CHR$(18);P0$;CHR$(30);N1$;P0$;CHR$(102);N2$;P0$;CHR$(174);
      N3$;P0$;CHR$(246);N4$;P1$;CHR$(62);N5$;P1$;CHR$(134);N6$;P1$;CHR$(206);N7$;
5790 GOSUB 87: FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT CHR$(18);P$;CHR$(Y1);CH
      R$(Y2);STRING$(2,140);: NEXT Y: GOSUB 88      'DECIMAL POINTS
5810 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
5820 GOSUB 87
5830 FOR Y=30 TO 462 STEP 72: GOSUB 85: LPRINT P$;CHR$(Y1);CHR$(Y2);N0$;: NEXT Y
      'SECOND "0" COLUMN
5835 GOSUB 6030      '4/72" LINE FEED
5840 RETURN
6000 '***** LINE FEEDS *****
6010 LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50);      '2/72"
6030 FOR LF=1 TO 4: LPRINT CHR$(27);CHR$(50);: NEXT LF      '4/72"
7000 '***** 5 X 11 CONDENSED PERPENDICULARS *****
7010 CD$=CHR$(30)+CHR$(27)+CHR$(20)+CHR$(18): NM$=CHR$(30)+CHR$(27)+CHR$(19)
7020 N1$=CD$+CHR$(156)+STRING$(7,136)+CHR$(140)+STRING$(2,136)+NM$
7030 N2$=CD$+CHR$(190)+STRING$(2,130)+CHR$(132)+CHR$(136)+CHR$(144)+STRING$(2,16
      0)+STRING$(2,162)+CHR$(156)+NM$
7040 N3$=CD$+CHR$(156)+STRING$(2,162)+CHR$(160)+CHR$(144)+CHR$(156)+CHR$(144)+CH
      R$(160)+STRING$(2,162)+CHR$(156)+NM$
7050 N4$=CD$+STRING$(3,144)+CHR$(190)+CHR$(146)+CHR$(146)+CHR$(148)+CHR$(152)+ST
      RING$(3,144)+NM$
7060 N5$=CD$+CHR$(156)+STRING$(2,162)+STRING$(3,160)+CHR$(158)+STRING$(3,130)+CH
      R$(190)+NM$
7070 N6$=CD$+CHR$(156)+STRING$(4,162)+CHR$(158)+STRING$(2,130)+STRING$(2,162)+CH
      R$(156)+NM$
7080 N7$=CD$+STRING$(3,132)+STRING$(2,136)+STRING$(2,144)+STRING$(2,160)+CHR$(16
      2)+CHR$(190)+NM$
7090 N8$=CD$+CHR$(156)+STRING$(3,162)+CHR$(148)+CHR$(156)+CHR$(148)+STRING$(3,16
      2)+CHR$(156)+NM$
7100 N9$=CD$+CHR$(156)+STRING$(2,162)+STRING$(2,160)+CHR$(188)+STRING$(4,162)+CH
      R$(156)+NM$
7110 N0$=CD$+CHR$(156)+STRING$(9,162)+CHR$(156)+NM$
7120 HM$=CD$+STRING$(4,128)+STRING$(3,190)+STRING$(4,128)+NM$
7130 NN$=CD$+CHR$(183)+STRING$(8,202)+CHR$(203)+CHR$(178)+NM$
7140 NE$=CD$+CHR$(247)+STRING$(8,162)+CHR$(179)+CHR$(162)+NM$
7150 NW$=CD$+CHR$(255)+STRING$(3,138)+CHR$(146)+CHR$(162)+STRING$(2,194)+CHR$(19
      7)+CHR$(235)+CHR$(178)+NM$
7160 NR$=CD$+CHR$(183)+STRING$(2,202)+STRING$(2,194)+CHR$(178)+STRING$(2,194)+CH
      R$(202)+CHR$(203)+CHR$(178)+NM$
7170 DP$=CHR$(18)+STRING$(7,255)      'BLACK SQUARE DATAPOINT
7180 GOTO 7200
7190 N1$=DP$: N2$=DP$: N3$=DP$: N4$=DP$: N5$=DP$: N6$=DP$: N7$=DP$: N8$=DP$: N9$
      =DP$: N0$=DP$: NE$=DP$: NW$=DP$: NR$=DP$
7200 RETURN
8000 '***** FILE NAMES AND NUMBER OF TEAMS *****
8010 IF FL$="AL01T060" OR FL$="NL01T061" THEN NT=8

```

```

8020 IF FL$="AL61T068" OR FL$="NL62T068" THEN NT=10
8030 IF FL$="ALE6976" OR FL$="ALW6976" OR FL$="NLE690N" OR FL$="NLW690N" THEN NT
    =6
8040 IF FL$="ALE770N" OR FL$="ALW770N" THEN NT=7
8050 IF FL$="ALEW6976" OR FL$="NLEW690N" THEN NT=12
8060 IF FL$="ALEW770N" THEN NT=14
8070 RETURN
9000 '***** PLOTTING WITHOUT CONNECTING LINES *****
9010 FOR XA=1 TO 4: GOSUB 82: LPRINT CHR$(27);CHR$(50);CHR$(27);CHR$(50);: NEXT
    XA
    'X-AXIS SEGMENTS
9099 RETURN

```

uously for over fifteen minutes. (Overheating of the printer's printhead is not so much of a danger as overheating of its positioning and bold-printing mechanisms.)

Point-and-range graphs. If we now add a set of data points running between the upper and lower limits of a range graph, we have what is called a point-and-range graph. This is a nice form for sweeping historical graphs in sports graphing that show how one team does in relation to the best and worst teams each year over a period of many years. In figure 9-9's point-and-range graph, the won-lost percentage of the Chicago White Sox is represented in the context of the whole range of won-lost percentages in the American League from 1901 through 1960. This is one of many graphs that can be produced by the PNTRANGE program when it accesses a databank containing the performance of all major-league baseball teams on nine different team statistics.

Rank-and-range graphs. A more informative product of the PNTRANGE program is the rank-and-range graph (figure 9-10), in which the highlighted team's performance is shown by numbers that stand for the team's rank in each year's final standings. How do we get these rank numbers? The same way we got the upper and lower limits—from the subscript of the array into which all teams' won-lost percentages have been sorted each season.

Rank-and-range graphs are not always entirely a form of line graph. Figure 9-10 shows a rank-and-range graph that is

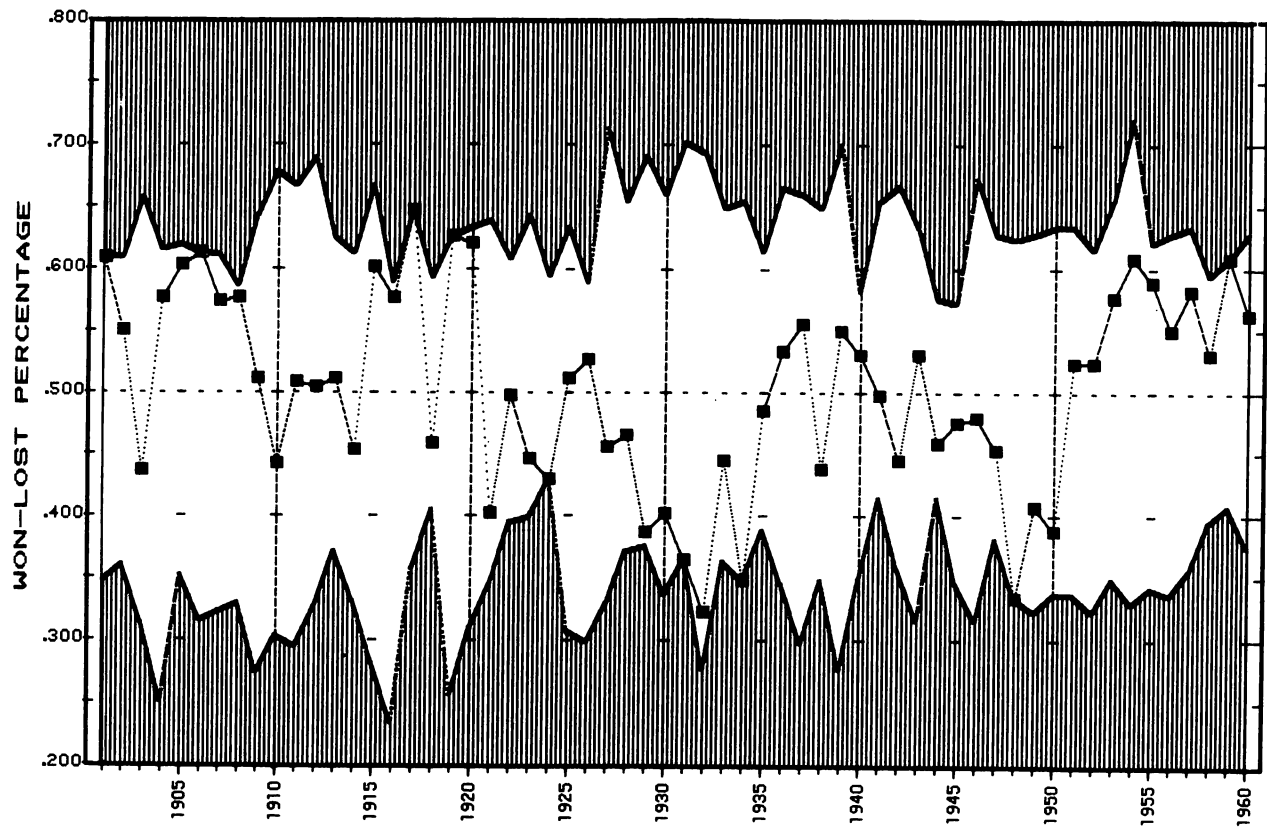


FIGURE 9-9. A point-and-range graph (PNTRANGE program). The winning percentage of the Chicago White Sox in relation to the American League's total range from 1901 to 1960 is displayed.

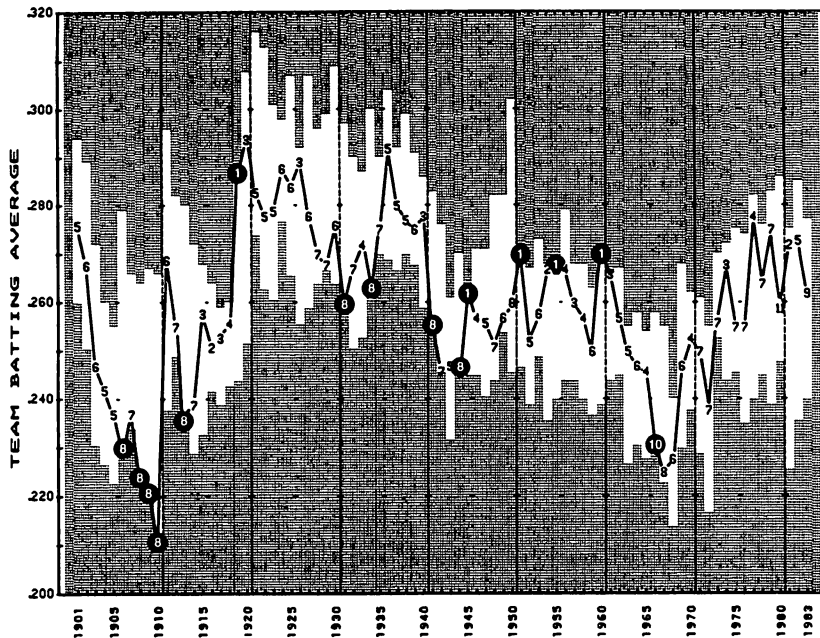


FIGURE 9-10. A rank-and-range graph using rank numbers as data points (the Chicago White Sox, dubbed the “Hitless Wonders,” vs. the American League range in team batting average).

partly a bar graph in that it uses shaded bars to show the range. This is a quick-and-dirty graph—quick because it takes only a fraction of the time to plot what the PNTRANGE form of rank-and-range graph requires, and dirty because the numerical data points for the highlighted team are connected by hand. Also, that team’s league-leading and league-trailing years are emphasized with dry-transfer white-on-black numbers.

The PNTRANGE program is designed to be used for simple range graphs, point-and-range graphs, and rank-and-range graphs. It is a long program mainly because it deals with nine different statistics and up to fourteen different teams. Subroutines 3000 and 4000, for example, are all concerned with sorting the teams and generating positioning values for the best, worst, and highlighted team of each year, but it takes many lines to do these things in slightly different ways for different statistical measures. Also, nine different ways of labeling the Y axis (subroutines 5000–5770) require

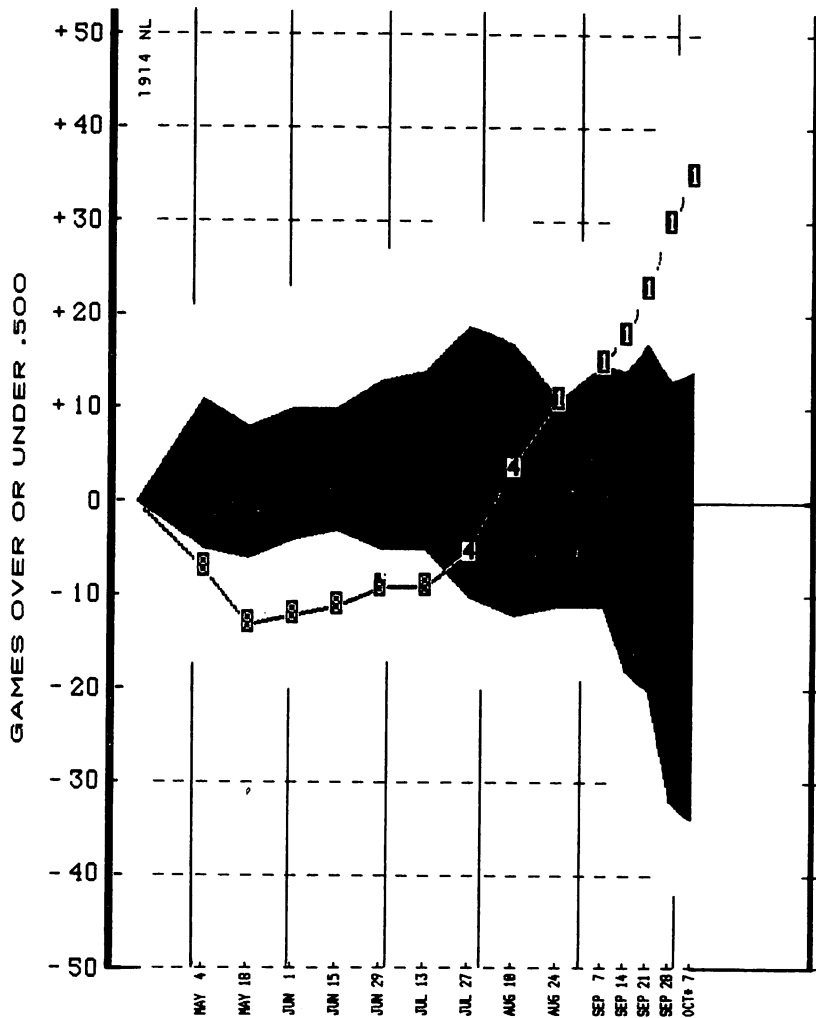


FIGURE 9-11. A variation of the rank-and-range graph that shows the Boston Braves' miraculous rise from last place to first in winning the National League pennant in 1914. The black area represents the range of the other seven teams in the league in won-lost percentage over the season. (Figure 12-7 in chapter 12 shows how printouts of this sort, having no X-axis lines, are used in multipanel graphs.)

many more lines than the sorting does. This is a lot of effort, but that will seem like a bargain when you realize that after slightly expanding the program to cover National as well as American League teams, it will generate at least 234 different point-and-range or rank-and-range graphs (nine measures

times twenty-six teams) and at least 18 different range graphs (nine measures times two leagues).

Black-white reversal can make the rank-and-range graph more elegant, especially if the highlighted team is plotted against the range of the *other* teams instead of all teams. See an example of this approach in figure 9-11. In the program that drew this (not listed), the black background is slowly drawn using a $\frac{3}{16}$ -inch line feed, a fresh ribbon in the printer, and a single pin of the printhead at a time. To avoid making any one of the printhead pins take more than its share of wear and tear, the program uses seven pins, one at a time, in a rotating order.

If the highlighted team is within the range of the other teams, the solid black printing is interrupted (by a shift to a blank code) to create the white-on-black connecting line and to make white space available for the printing (in black) of the highlighted team's rank. If the team is above or below the range of the others, its position is shown by a black rectangle containing a white rank number, and its connecting lines are black. Another example of this kind of graph appears in the final chapter (figure 12-7).

Tandem Bar-and-Line Graphs

Bar and line graphs in combination can pack a lot of information into fairly small spaces. They can also help interpret information by revealing relationships, and they can often be efficient in having two sets of data shown with a common X axis.

Back to baseball again for an example (figure 9-12). In this case a bar graph is used to show the batting output of one player (Ted Williams, when he ended the season with a .406 batting average in 1941) on a game-by-game basis in the lower part of the graph. Meanwhile, the player's batting average (hits divided by times at bat) is displayed by a line graph in the upper part.

For data input the program that produced this double display requires the date, location, a four-digit number from the day's box score (giving at-bats, runs, hits, and runs batted in), and an indication of how many singles, doubles, triples, and home runs the player hit that day. The latter four are con-

TED WILLIAMS: GAME BY GAME IN 1941

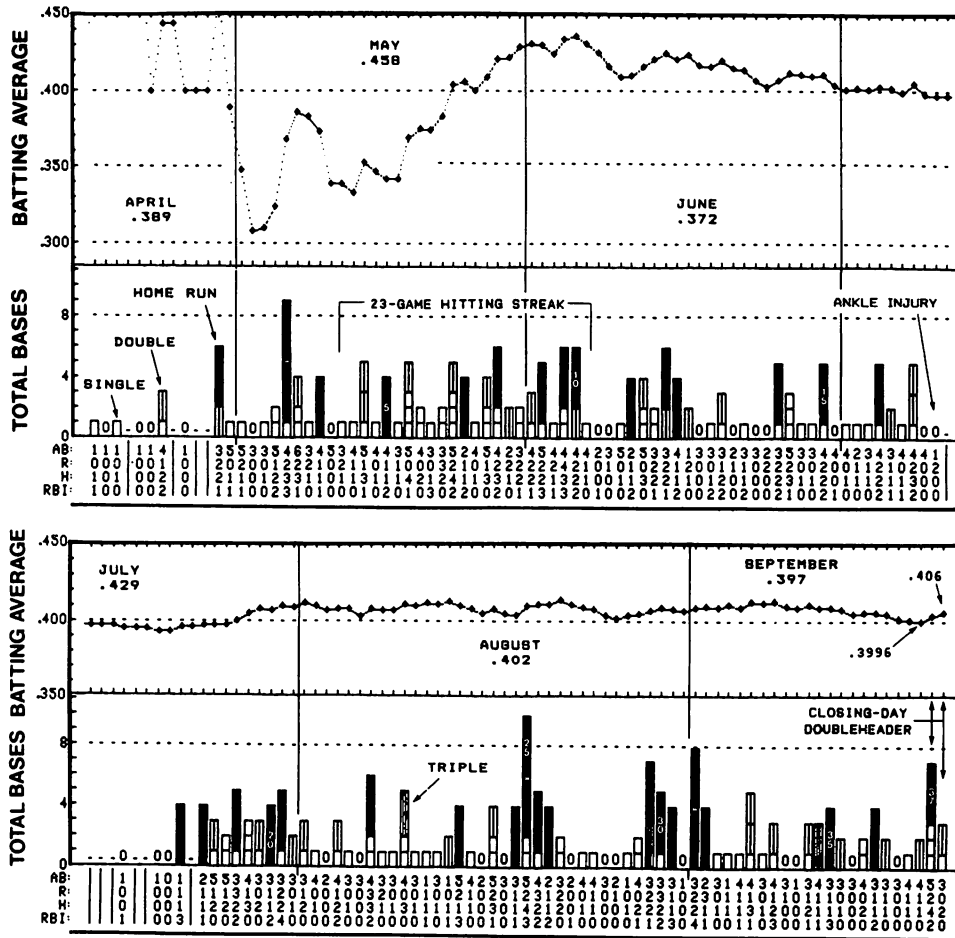


FIGURE 9-12. A combination bar-and-line graph showing Ted Williams's daily batting output in 1941. (Adapted with permission of Macmillan Publishing Company from *Boston Red Sox* by Henry Berry and Harold Berry, graphics by John W. Davenport. Copyright © 1984 by Macmillan Publishing Company, a division of Macmillan, Inc.)

verted to rectangles of differing size that become the components of the total-bases measure in the lower portion of the graph (as in the segmented bar graph, figure 8-4). The date and location can be printed between the bar and line graphs, or if that strikes you as being too cluttered, you can put those at the bottom or top or leave them out entirely.

As with some of our recent examples of line graphs, this tandem graph was produced almost entirely by the computer

and printer, with only a few manual finishing touches (in the labeling in this case, not the connecting lines). It may make the programmer feel proud to avoid those touches, but they usually go a long way toward improving the appearance and clarity of the graph. We'll keep running into this issue.

... And More Graphs

THERE are several more types of graphs that can't be classed as bar or line graphs. Among others, these include scatterplots, ladder graphs, and pie charts.

Scatterplots

With all the talk lately about how the outcome of the Super Bowl has forecast the direction of the stock market almost perfectly over the past eighteen years, we ought to take a closer look. If you haven't heard about this, here's a recap: In 1970, a merger took place in which the teams of the American Football League (AFL) became part of the National Football League (NFL). Prior to the merger, four Super Bowls were played between the American and National Football League champions; subsequently (through 1984), fourteen Super Bowls have been played between the top two teams of the expanded NFL. Except in 1984, every year in which a team that was originally in the NFL has won the Super Bowl, the Standard & Poor's five-hundred-stock average, measured from January 1 to December 31, has gone up. Conversely, in every year that the winner was a team originally from the AFL, the same stock average has gone down. And even in 1984, when the Los Angeles Raiders, a former AFL team, won by a big margin, there was only a meager 1.04 rise in the S&P index, as if the Raiders' victory had amputated the "second leg of the bull market" that most investors had expected to develop in 1984.

If this has all happened by chance, it seems to be a statistical wonder. On the surface, it appears that through 1983 the Super Bowl outcome predicted the stock market change for seventeen years in a row. In random guessing at a two-outcome event, the likelihood of being correct seventeen times out of seventeen is one chance in 131,072, and being correct seventeen times out of eighteen (including 1984) by chance also has a tiny probability—one in 13,107. But it isn't this simple.

Table 10-1 has all the Super Bowl scores and stock market figures. One thing to notice right away is that in four Super Bowls (1971, 1975, 1979, and 1980) both teams were former NFL teams. And while the stock index went up in all four of those years, the cleanest way to calculate the likelihood that the overall relationship is due to chance is to throw out the data for those years. That leaves us with thirteen out of thirteen through 1983 and thirteen out of fourteen through 1984, amounting to one chance in 8,192 through 1983 and one chance in 1,072 if we include 1984—both of which are still very small probabilities.

A more interesting way of examining this relationship is to run a correlation and look at the data in a scatterplot. We can do this by plotting the point spread in each Super Bowl's final score (call that X) and that year's rise or fall in the S&P market average (call that Y) together in the same chart.

Figure 10-1 is a scatterplot of these data that was produced by program 10-1, PEARSONR. Besides drawing the graph, this program calculates the Pearson correlation coefficient (r), which indicates the strength of association between the X and Y variables. The Pearson r can vary from -1.0 (perfect negative correlation) through 0 (no correlation) to $+1.0$ (perfect positive correlation). In the scatter graph the point spread (defined by subtracting the AFL score from the NFL score) and stock market index are plotted for each year. The usual practice of plotting filled circles or other typical data points has been abandoned in favor of a more informative method of using two-digit prints identifying the years. Notice that the years in which two NFL teams played each other are not included and that 1984 is.

Here we see that the correlation between Super Bowl outcomes and stock-market changes is $+.55$. This also is unlikely to be due to chance: With fourteen pairs of scores, the

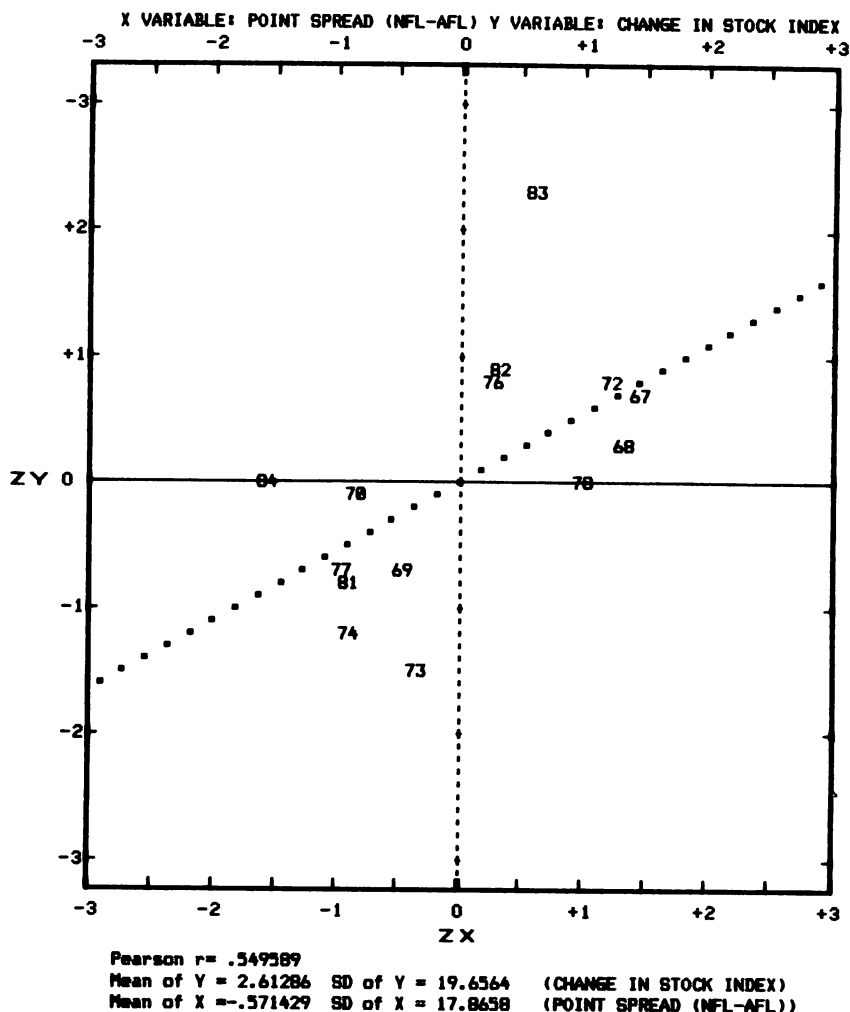


FIGURE 10-1. A scatterplot showing how Super Bowl outcomes have related to annual changes in the stock market from 1967 to 1984 (PEARSONR program). Also see table 10-1.

likelihood of the true value of r being 0 is only about four in a hundred. (Before 1984, the correlation was $+ .60$ with about the same probability.) Compared to that seventeen-out-of-seventeen probability we started with, though, this is pretty modest, and those who insist on a 1% level of confidence would say that the correlation is not significantly different from zero. But remembering the $1/1024$ for thirteen out of four-

TABLE 10-1
Super Bowl Outcomes and Changes in the Standard & Poor's Stock Market Index

<u>Year</u>	<u>Score</u>	<u>Point Spread</u> <u>(NFL-AFL)</u>	<u>Market Change</u> <u>(+=up, -=down)</u>
1967	Green Bay (N) 35, Kansas City (A) 10	+25	+15.73
1968	Green Bay (N) 33, Oakland (A) 14	+19	+ 7.62
1969	N. Y. Jets (A) 16, Baltimore (N) 7	- 9	-11.63
1970	Kansas City (A) 23, Minnesota (N) 7	-16	- 1.02
1971	Baltimore (N) 16, Dallas (N) 13	+ 3*	+10.69
1972	Dallas (N) 24, Miami (A) 3	+21	+16.77
1973	Miami (A) 14, Washington (N) 7	- 7	-27.75
1974	Miami (A) 24, Minnesota (N) 7	-17	-21.80
1975	Pittsburgh (N) 16, Minnesota (N) 6	+10*	+20.84
1976	Pittsburgh (N) 21, Dallas (N) 17	+ 4	+16.96
1977	Oakland (A) 32, Minnesota (N) 14	-18	-12.83
1978	Dallas (N) 27, Denver (A) 10	+17	+ 1.38
1979	Pittsburgh (N) 35, Dallas (N) 31	+ 4*	+11.79
1980	Pittsburgh (N) 31, Los Angeles Rams (N) 19	+12*	+28.06
1981	Oakland (A) 27, Philadelphia (N) 10	-17	-13.11
1982	San Francisco (N) 26, Cincinnati (A) 21	+ 5	+18.56
1983	Washington (N) 27, Miami (A) 17	+10	+46.67
1984	L. A. Raiders (A) 38, Washington (N) 9	-29	+ 1.04
1985	San Francisco (N) 38, Miami (A) 16	+22	?

* Both teams were former NFL teams. Point spread = winner - loser.

teen, logic is on the side of those who conclude that the Super Bowl and stock market outcomes are really related.

Why these two variables should be related is, of course, obvious. It is well known that most investors in stocks are older, more conservative types who grew up and became football fans before there even was an AFL. Naturally, they developed deep loyalties to old NFL teams like the Washington Redskins, Pittsburgh Steelers, and Green Bay Packers. And just as naturally, whenever in more recent years one of these teams won the Super Bowl, it made them happy and put them in a more optimistic frame of mind, whereas they became growling bears when an old AFL team won. The remarkable thing is that their bullish or bearish moods persisted for eleven more months each year, driving the market prices up or down accordingly.

PROGRAM 10-1. PEARSONR. Correlation scatterplot for Pearson r situations.

```

10 'PROGRAM 10-1: "PEARSONR" -- CORRELATION SCATTERPLOT
15 'Copyright (c) 1985 by John Warner Davenport
18 'FOR TRS-80 MODEL III (W/ BOOTHE'S DRIVER) OR MODEL 4 AND TRS-80 PRINTERS
20 CLEAR 3000
30 DIM TAG$(200),ZX(200),ZY(200),C$(200),MF$(200)
40 INPUT "NAME OF X VARIABLE";XV$
50 INPUT "NAME OF Y VARIABLE";YV$
60 GOSUB 1010 'MASTER RESET (PRINTER)
70 INPUT "IS YOUR COMPUTER A TRS-80 MODEL III (Y OR N)";CQ$
80 INPUT "DO YOU WANT THE REGRESSION LINE PLOTTED (Y OR N)";RQ$
90 READ TAG$,X,Y
92 IF TAG$="71" OR TAG$="75" OR TAG$="79" OR TAG$="80" THEN GOTO 90
95 IF TAG$="N" THEN GOTO 90
100 IF TAG$="DONE" THEN GOTO 140
110 XX=X*X: YY=Y*Y: XY=X*Y: N=N+1
120 TX=TX+X: TY=TY+Y: QX=QX+XX: QY=QY+YY: SP=SP+XY
130 GOTO 90
140 PRINT "END OF FIRST DATA INPUT. N=";N;" PAIRS OF SCORES."
145 REM -- MODEL III USERS: CHANGE ^s TO LEFT-BRACKET SYMBOLS FOR EXPONENTIATIO
    N IN LINES 150 AND 190
150 VX=(N*QX-(TX^2))/(N*(N-1)):VY=(N*QY-(TY^2))/(N*(N-1)) 'VARIANCES
160 SX=SQR(VX): SY=SQR(VY) 'STANDARD DEVIATIONS OF X & Y
170 MX=TX/N: MY=TY/N 'MEANS OF X & Y
180 PRINT "MEAN OF X=";MX;" S.D.OF X=";SX;" MEAN OF Y=";MY;" S.D. OF Y=";SY
190 RR=(N*SP-TX*TY)/SQR((N*QX-TX^2)*(N*QY-TY^2)) 'RAW-SCORE CALCULATION OF PE
    ARSON r
200 PRINT "PEARSON r (FROM RAW-SCORE FORMULA) =" ;RR
210 RESTORE
220 FOR I=1 TO N
230 READ TAG$,X,Y
232 IF TAG$="71" OR TAG$="75" OR TAG$="79" OR TAG$="80" THEN GOTO 230

```

```

235 IF TAG$="N" THEN GOTO 230
237 IF TAG$="DONE" THEN GOTO 320
240 ZX(I)=(X-MX)/SX: ZY(I)=(Y-MY)/SY: ZZ=ZX(I)*ZY(I): TAG$(I)=TAG$
250 SCP=SCP+ZZ      'SUMMATION OF CROSS-PRODUCTS (ZX(I)*ZY(I))
260 R=SCP/N        'CALCULATION OF PEARSON r FROM Z -SCORES
270 ZY(I)=INT(100*ZY(I)+500)      'CONVERT ZY(I) TO 200-800 SCALE
280 ZX(I)=INT(100*ZX(I)+500)      'CONVERT ZX(I) TO 200-800 SCALE
290 C$(I)=STR$(ZY(I))+ " "+TAG$(I)+STR$(ZX(I))      'FORM STRING ARRAY
300 PRINT "C$(I)=";C$(I)
310 NEXT I
320 PRINT "END OF DATA INPUT. R=";R;" "; "LAST C$(I)=";C$(N)
330 IF CQ$="N" OR CQ$="NO" THEN GOSUB 10000: GOTO 350 'SHELL-METZNER SORT
340 NZ=N: CMD "O", NZ, C$(I)      'RAPID SORTING OF C$(I) ON MODEL III
350 PRINT "ORDERED Y VALUES:"
360 FOR I=N-1 TO 0 STEP -1
370 PRINT TAB(5);C$(I),I;
380 NEXT I
390 LPRINT TAB(12);"X VARIABLE: ";XV$;" Y VARIABLE: ";YV$;
400 GOSUB 9990 : GOSUB 9990
410 GOSUB 6000      'NUMBERING OF TOP X AXIS
420 GOSUB 9990 : GOSUB 9990
430 GOSUB 3010      'TOP X AXIS, HASHMARKS
450 FOR J=820 TO 180 STEP -10
455 JM=J+10
460 I=N-1: IF I=0 THEN GOTO 490
470 ZY(I)=VAL(LEFT$(C$(I),4)):ZX(I)=VAL(RIGHT$(C$(I),4)): TAG$(I)=MID$(C$(I),6,2)
)
472 PRINT "J=";J,"JM=";JM,"ZY(I)=";ZY(I),"YEAR=";TAG$(I),"ZX(I)=";ZX(I)
480 IF ZY(I)<=JM AND ZY(I)>J THEN GOSUB 8500 : N=N-1: GOTO 460 'PRINTING OF DATA POINT
TAPOINT
490 IF RQ$="Y" THEN K=(J-490)/100: RL=K/R: B=70*(RL+3)+45
500 IF B>45 AND B<463 THEN GOSUB 9500 'PRINTING OF POINT ON THE REGRESSION LINE
NE
520 GOSUB 9990 : GOSUB 9000 : GOSUB 8010 'LINE FEED, L & R Y AXES
530 IF J=180 THEN GOSUB 4000 : GOSUB 9990 : GOSUB 9990 : GOSUB 6000 : GOSUB 9990
: GOSUB 2000 : GOTO 980
550 NEXT J
980 PRINT "JOB DONE"
990 END
1000 '***** MASTER RESET (PRINTER) *****
1010 LPRINT CHR$(30);CHR$(27);CHR$(23);CHR$(27);CHR$(15);CHR$(27);CHR$(32);
1020 GR$=CHR$(18)+CHR$(27)+CHR$(16)
1030 RETURN
2000 '***** STATISTICAL REPORT *****
2010 GOSUB 9990
2020 LPRINT CHR$(30);TAB(11);"Pearson r=";R
2030 LPRINT TAB(10);"Mean of Y ="MY;" SD of Y ="SY;" ("YV$;")"
2040 LPRINT TAB(10);"Mean of X ="MX;" SD of X ="SX;" ("XV$;")"
2050 RETURN
3000 '***** PRINTING OF TOP X AXIS *****
3010 LPRINT GR$;CHR$(0);CHR$(44);STRING$(2,255);STRING$(209,131);STRING$(3,255);
STRING$(208,131);STRING$(2,255);
3020 FOR A=430 TO 45 STEP -35: QA=FIX(A/256): AA=FIX(A-256*QA): LPRINT GR$;CHR$(
QA);CHR$(AA);CHR$(255);: NEXT A 'HASHMARKS

```

```

3030 RETURN
4000 '***** PRINTING OF BOTTOM X AXIS*****
4010 LPRINT GR$;CHR$(0);CHR$(44);STRING$(2,255);STRING$(209,224);STRING$(3,255);
      STRING$(208,224);STRING$(2,255);
4020 FOR A=430 TO 45 STEP -35: QA=FIX(A/256): AA=FIX(A-256*QA): LPRINT GR$;CHR$(
      QA);CHR$(AA);CHR$(255);: NEXT A
4030 RETURN
5000 '+---+---+--- POSITIVE & NEGATIVE NUMBERS ---+---+---+
5010 PL$=CHR$(18)+STRING$(2,136)+CHR$(190)+STRING$(2,136)+STRING$(2,128)
      '""
5020 MI$=STRING$(5,136)+STRING$(2,128)
      '""
5030 P3$=PL$+CHR$(162)+CHR$(193)+STRING$(2,201)+CHR$(182) ' "+3"
5040 M3$=MI$+CHR$(162)+CHR$(193)+STRING$(2,201)+CHR$(182) ' "-3"
5050 P2$=PL$+CHR$(226)+CHR$(209)+STRING$(2,201)+CHR$(198) ' "+2"
5060 M2$=MI$+CHR$(226)+CHR$(209)+STRING$(2,201)+CHR$(198) ' "-2"
5070 P1$=PL$+CHR$(128)+CHR$(194)+CHR$(255)+CHR$(192) ' "+1"
5080 M1$=MI$+CHR$(128)+CHR$(194)+CHR$(255)+CHR$(192) ' "-1"
5090 ZE$=CHR$(190)+STRING$(3,193)+CHR$(190)
      ' "0"
5100 RETURN
6000 '***** NUMBERING OF X AXIS *****
6010 GOSUB 5000
6020 LPRINT GR$;CHR$(0);CHR$(39);M3$;GR$;CHR$(0);CHR$(109);CHR$(0);M2$;GR$;CHR$(
      0);CHR$(179);M1$;GR$;CHR$(0);CHR$(254);ZE$;
6030 LPRINT GR$;CHR$(1);CHR$(64);P1$;GR$;CHR$(1);CHR$(134);P2$;GR$;CHR$(1);CHR$(
      204);P3$;
6040 IF J=180 THEN GOSUB 9990 : GOSUB 9990 : LPRINT GR$;CHR$(0);CHR$(246);CHR$(3
      0);CHR$(27);CHR$(14);"ZX";CHR$(27);CHR$(15);
6050 RETURN
7000 '***** LABELING, HASHMARKS OF Y AXES *****
7010 LPRINT GR$;CHR$(0);CHR$(30);N$;GR$;CHR$(0);CHR$(44);STRING$(2,255);STRING$(
      4,136);
7020 LPRINT GR$;CHR$(0);CHR$(255);CHR$(136);CHR$(190);CHR$(136);
7030 LPRINT GR$;CHR$(1);CHR$(206);STRING$(4,136);STRING$(2,255);
7040 RETURN
7500 '***** HORIZONTAL LINE AT Y=0 *****
7510 LPRINT GR$;CHR$(0);CHR$(0);CHR$(30);CHR$(27);CHR$(14);"ZY";CHR$(27);CHR$(15
      );GR$;CHR$(0);CHR$(45);STRING$(210,136);STRING$(211,136);: RETURN
8000 '***** LEFT AND RIGHT Y AXES *****
8010 LPRINT GR$;CHR$(0);CHR$(44);STRING$(2,255);GR$;CHR$(1);CHR$(0);CHR$(156);GR
      $;CHR$(1);CHR$(210);STRING$(2,255);
8020 RETURN
8500 '***** PRINTING OF DATAPOINTS *****
8510 YR$=MID$(C$(I),6,2)
8520 DF$=CHR$(18)+CHR$(156)+CHR$(190)+STRING$(3,255)+CHR$(190)+CHR$(156)
8530 DO$=CHR$(18)+CHR$(156)+CHR$(162)+STRING$(3,193)+CHR$(162)+CHR$(156)
8540 ZX(I)=(ZX(I)-500)/100: A=70*(ZX(I)+3)+45: QA=FIX(A/256): AA=FIX(A-256*QA)
8545 IF AA=9 THEN AA=8
8550 IF TAG$(I)="M" THEN LPRINT GR$;CHR$(QA);CHR$(AA);CHR$(30);CHR$(8);CHR$(6);D
      F$;: GOTO 8580
8560 IF TAG$(I)="F" THEN LPRINT GR$;CHR$(QA);CHR$(AA);CHR$(30);CHR$(8);CHR$(6);D
      O$;: GOTO 8580
8570 LPRINT GR$;CHR$(QA);CHR$(AA);CHR$(30);CHR$(8);CHR$(6);YR$;
8580 RETURN
9000 '***** Y-AXIS NUMBERING *****

```

```

9010 IF J=800 THEN N$=M3$: GOSUB 7010 : RETURN
9020 IF J=700 THEN N$=P2$: GOSUB 7010 : RETURN
9030 IF J=600 THEN N$=P1$: GOSUB 7010 : RETURN
9040 IF J=500 THEN N$=ZE$: GOSUB 7010 : GOSUB 7510 : RETURN
9050 IF J=400 THEN N$=M1$: GOSUB 7010 : RETURN
9060 IF J=300 THEN N$=M2$: GOSUB 7010 : RETURN
9070 IF J=200 THEN N$=M3$: GOSUB 7010 : RETURN
9080 RETURN
9500 '***** PRINTING OF REGRESSION LINE *****
9510 QB=FIX(B/256): BB=FIX(B-256*QB)
9520 LPRINT GR$;CHR$(QB);CHR$(BB);CHR$(30);CHR$(8);CHR$(1);CHR$(18);STRING$(3,15
6);
9530 RETURN
9980 '***** GRAPHIC LINE FEED W/ CARRIAGE RETURN *****
9990 LPRINT CHR$(18);CHR$(13);: RETURN
10000 '***** SHELL-METZNER SORT OF C$(I) *****
10008 'MODEL 4: FOR 2-SEC. SORT, USE SMITH'S PROGRAM (80 MICRO, MARCH 1985)
10010 PRINT "SORTING NOW (SHELL-METZNER)"
10020 P=0
10030 Q=N
10040 Q=INT(Q/2)
10050 IF Q=0 THEN GOTO 10230
10060 P=P+1: PRINT "PASS";P;
10070 FOR BL=0 TO Q-1
10080   I=BL
10090   J=BL+Q
10100   SW=0
10110   IF C$(I)<=C$(J) THEN GOTO 10160
10120   SW=1
10130   B$=C$(I)
10140   C$(I)=C$(J)
10150   C$(J)=B$
10160   I=J
10170   J=J+Q
10180   IF J<N THEN GOTO 10110
10190   IF SW=0 THEN GOTO 10210
10200 GOTO 10080
10210 NEXT BL
10220 GOTO 10040
10230 PRINT "SORTING FINISHED"
10240 RETURN
14000 REM DATA --YEAR,POINT SPREAD,S&P CHANGE
14010 DATA 67,25,15.73
14020 DATA 68,23,7.62
14030 DATA 69,-9,-11.63
14040 DATA 70,-16,-1.02
14050 DATA 71,3,10.69
14060 DATA 72,21,16.77
14070 DATA 73,-7,-27.75
14080 DATA 74,-17,-21.81
14090 DATA 75,10,20.84
14100 DATA 76,4,16.96
14110 DATA 77,-18,-12.83
14120 DATA 78,17,1.38

```

```

14130 DATA 79,4,11.79
14140 DATA 80,12,28.06
14150 DATA 81,-17,-13.11
14160 DATA 82,5,18.56
14170 DATA 83,10,46.67
14175 DATA 84,-29,1.40
14180 DATA DONE,9999,9999
15000 'TO ELIMINATE NFL-VS.-NFL YEARS FROM THE DATA, INSERT LINE 92: 'IF TAG$="7
1" OR TAG$="75" OR TAG$="79" OR TAG$="80" THEN GOTO 90' AND THE SAME STATE
MENT IN LINE 232 EXCEPT FOR ENDING WITH 'GOTO 230'

```

The PEARSONR program is designed to be useful for virtually any correlation situation in which the two variables are continuous dimensions rather than categories (such as male vs. female). For proper interpretations of the Pearson r , the relationship between the X and Y variables should be a straight-line (linear) one rather than a curved (nonlinear) one, and since the program plots a scatter graph, you can tell pretty easily which of these is the case.

The program converts all of the X and Y scores to standard scores, called ZX and ZY scores, on scales that run from -3 to $+3$. Zero on one of these scales represents the arithmetic mean, and about two-thirds of the scores of most variables will fall between -1 and $+1$ on a standard-score scale.

In order to convert any raw score (X or Y) to a ZX or ZY score, you have to calculate the mean (MX or MY) of the raw scores and also get an index of how much they vary, known as the standard deviation (SX or SY). Then the conversion is simple: The difference between any raw score and its mean is divided by the standard deviation. In symbols, this amounts to $ZX = (X - MX)/SX$ for X scores and $ZY = (Y - MY)/SY$ for Y scores.

The first part of the PEARSONR program is concerned with these calculations. It reads all the data in the DATA statements (up to two hundred pairs of X and Y scores) a first time, and from this sweep it determines how many pairs of scores there are (N). While doing that, it adds up various ingredients needed for calculating means, standard deviations, and the correlation coefficient, r , itself. When the last data item is disposed of, these calculations are made and reported on the computer's screen (lines 140–200). Then, in line 210, the READ-DATA pointer is reset to the starting point by the **RESTORE** command, and the DATA statements

are read a second time. In this second run-through, the X and Y scores are converted to ZX and ZY scores, and their cross-products (the result of multiplying each ZX score by its paired ZY score) are cumulated. These cross-products (ZZ), when summed and divided by N, as in lines 250 and 260, provide another calculation of the correlation coefficient as a check against its earlier calculation from the raw scores.

The stage is set for drawing the scatterplot in line 290 by the formation of a string array, C\$(I), to permit us to sort the data in order of ZY(I) values without breaking up the ZY-ZX pairings or losing the identification (TAG\$). This is done by concatenating ZY(I) with TAG\$(I) and ZX(I), after each ZY(I) value has been multiplied by 100 and a constant of 500 has been added, in order to make all ZY(I) values the same length. The sorting is done in two seconds by CMD "O" on the TRS-80 Model III, but if that's not available, the Shell-Metzner sort in subroutine 10000 will order the C\$(I) values in a reasonable period of time.

Once sorted, the C\$(I) array is ready for plotting. After drawing the upper X axis, the program starts at the now-converted ZY(I) value of 820 and steps down ten units at a time hunting for ZY(I) values in the left portion of the C\$(I) strings. With each step small segments of the left and right Y axes are drawn, followed by a line feed. Whenever a value of ZY(I) is found that falls within the ten-unit window (for example, between 740 and 730), the data point for the ZX(I) value at the right end of the C\$(I) is positioned on the X scale and printed. The data point may be a conventional symbol such as a filled or open circle, or it may be a two-digit number that identifies a year, as in figure 10-1, in the middle (TAG\$) part of C\$(I), depending on what is specified at the end of line 8570 (YR\$, DF\$, or DO\$).

At certain points in the down-stepping (after every ten steps), the Y axis is numbered and labeled with hash marks, and at the 500 point the Y-axis zero line is drawn across the graph. When all the points have been plotted and the 180 point is reached, the lower X axis is drawn and numbered. After that the value of the correlation (r) is printed, along with the means (MX and MY) and standard deviations (SX and SY) in terms of their original raw-score scales. With this information, the standard scores can easily be translated back into raw scores: A ZX of zero equals MX and a ZX of +1 or

-1 equals the standard deviation, SX . The same holds for ZY scores as they are numbered on the Y axis (that is, from -3 to $+3$, not on the 200–800 scale): set ZY of zero equal to MY and ZY s of $+1$ or -1 equal to SY . In our example, MX is 2.9 and SX is 14.7, so ZX scores of -3 , -2 , -1 , 0 , $+1$, $+2$, and $+3$ translate into -41.2 , -26.5 , -11.8 , 2.9 (the mean), 17.6 , 32.3 , and 47.0 in terms of the point spreads of Super Bowl outcomes. And with MY equal to 6.3 and SY equal to 19.2, the seven values for the Y variable become -51.3 , -32.1 , -2.9 , 6.3 , 25.5 , 44.7 , and 63.9 .

The line for predicting Y values from X values, called the regression line in the textbooks, can be calculated directly from the correlation coefficient and can also be plotted as the data points are being plotted. The equation for a regression line is $Y = bX + c$, with b standing for the slope of the line and c representing where (on the Y scale) the line crosses the vertical at $ZX=0$. Since the PEARSONR program converts all raw scores to ZX or ZY standard scores, c will always be zero and b will equal the correlation coefficient. This means that the line will always pass through the point where the horizontal and vertical lines at $ZX=0$ and $ZY=0$ cross in the middle of the scatterplot. It also means that the slope value, b , can range from -1 to $+1$, just like a Pearson r value.

In our example, the regression line tells us that for every unit change in ZX there is .55 of a unit change in ZY . From the standard deviations printed at the bottom of the graph, we can translate this to mean that, on the average, for every ten points in the direction of an old NFL team winning the Super Bowl, there is a 6.1-point increase in the Standard & Poor's stock market average for the year.

The PEARSONR program can handle up to 200 pairs of X and Y scores, and even more than that if you increase the subscripts (in parentheses) in the DIM statement near the beginning of the program. It can also plot two or more different data points in the scatterplot to represent different items in TAG\$. These are advantages over some of the commercial software packages for scatterplots, which may limit you to one hundred pairs of scores and a single data point symbol. Figure 10-2 illustrates how pairs of scores for men may be distinguished from those for women by filled and unfilled circles, respectively.

For both sexes, this second scatterplot shows the correla-

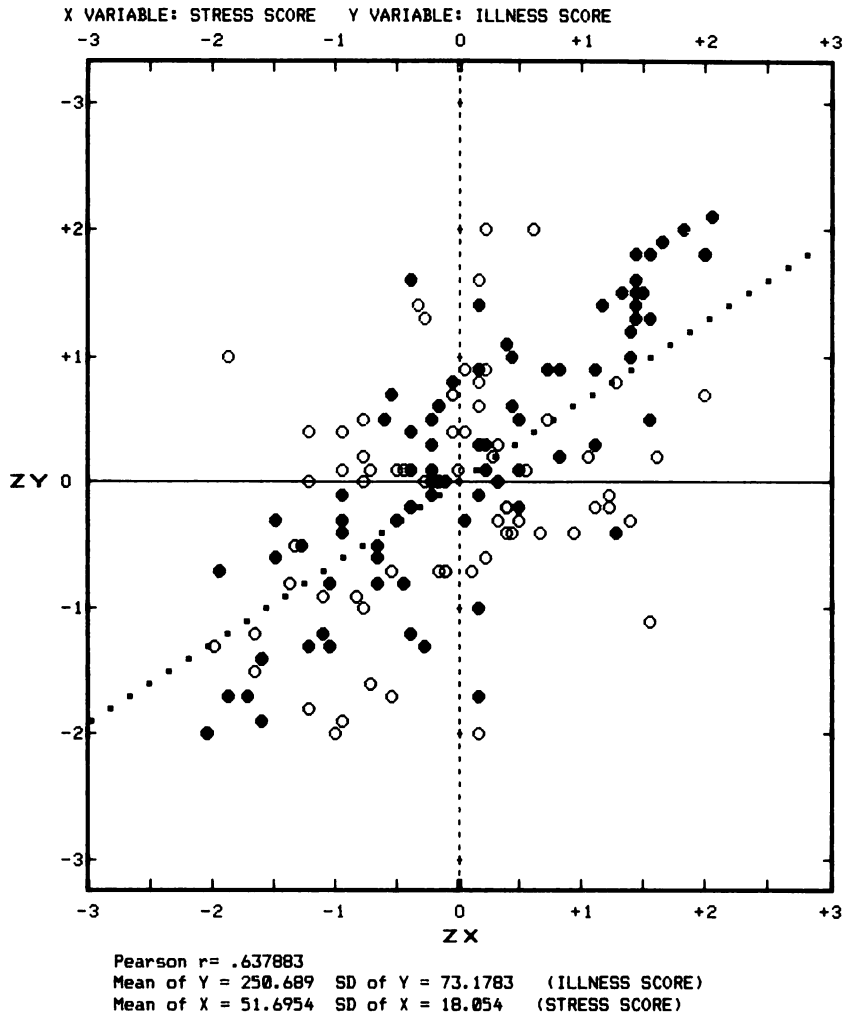


FIGURE 10-2. A scatterplot showing the relationship between psychological stress and physical health (fictional data) for a combined group of men and women (PEARSONR program).

tion between psychological stress and physical illness (the data are purely fictional). Stress is the X variable; the raw stress scores can range from 0 to 100. Illness is the Y variable, on a scale running from 100 to 400. When the men's and women's pairs of scores are combined into a single set of data, the correlation coefficient comes out $+.64$ (see the unrounded r value at the bottom of figure 10-2).

The DATA listings in this example will, of course, be different from those in our football example, but the format can

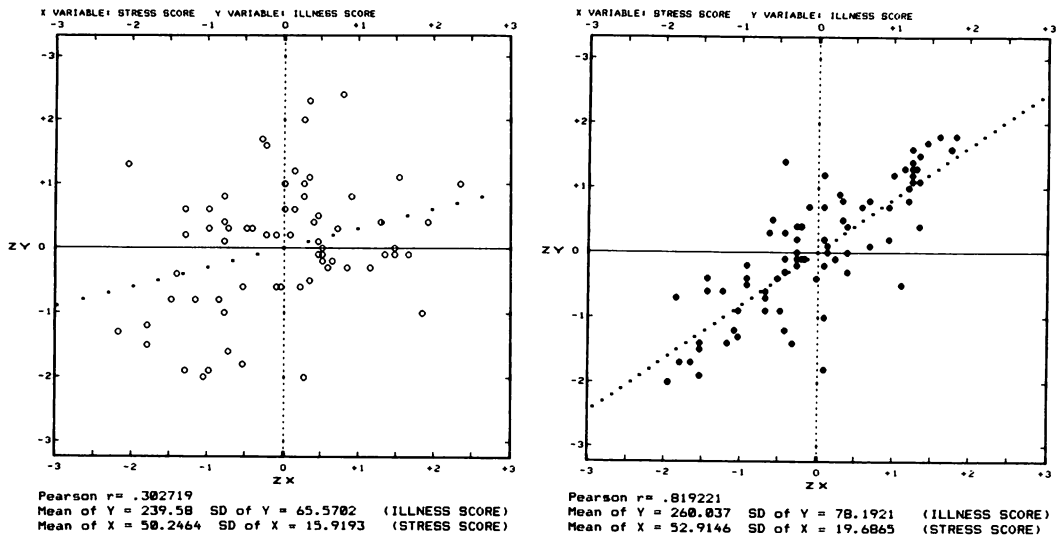


FIGURE 10-3. Stress-and-illness relationships plotted separately for women (left) and men (right) by the PEARSONR program.

be similar—label, X score, and Y score. Now, instead of year, point spread, and stock market change, we have M or F, stress score, and illness score. Using TAG\$(M or F) for identifying males and females enables us to plot their data points in different ways and even in different scatterplots.

If you look closely at the filled and open circles in figure 10-2's scatterplot, though, you'll end up wondering how different the relation between stress and illness may be for men than for women—the mens' data points seem to show more of a distinct trend than the women's. Perhaps that correlation of .64 is a deceiving kind of rough average of two things that shouldn't be combined.

In situations like this, to get at the undistorted truth, we should look at separate scatterplots for the men and women and calculate separate r values as well. To do this for the men, simply change lines 95 and 235 in the PEARSONR program to

```

95 IF TAG$="F" THEN GOTO 90
235 IF TAG$="F" THEN GOTO 230

```

and change line 470's TAG\$(I)=MID\$(C\$(I),6,2) to TAG\$(I)=MID\$(C\$(I),5,1). For the women's calculations and plotting, change "F" to "M".

Figure 10-3 gives us the separated results, and see how much clearer they have become! The correlation between

stress and illness turns out to be $+ .82$ for the men and only $+ .30$ for the women. The scatterplots tell the story, by making it easy to see how much closer to a random dot pattern (i.e., zero correlation) the data points for the women are. PEARSONR may be a bit long, but it's a handy program.

Rank correlations. Fairly often we have to start with scores that come only in the form of ranks. For example, we are told that the states having the highest murder rates in 1982 were Alaska, Texas, Louisiana, Mississippi, Nevada, Florida, Georgia, New Mexico, New York, and California, in that order (these are actual rankings from the FBI's Uniform Crime Reports, adjusted for population). In rates of auto theft, the same states ranked (among themselves and not all fifty states) 6th, 2nd, 5th, 9th, 7th, 10th, 4th, 8th, 1st, and 3rd, respectively.

If we wanted to find out what the correlation between murder and auto theft rates was for just these few states, we couldn't use the Pearson r because that is not designed to handle ranks. Another kind of correlation statistic, called the Spearman rank-order correlation coefficient (r_s for short), will handle the job easily. You simply find the difference between each state's two ranks (i.e., rank in murder and rank in auto theft), square the differences, add these ten squares, and plug their sum (call it SSQ) into a formula. In BASIC that formula reads $RS = 1 - (6 * SSQ / (N^3 - N))$, N being the number of pairs of ranks. In our crime example the r_s comes out to be $-.10$, which is too close to zero to be significant.

There's not much gained by plotting such small numbers of pairs, but when N is greater, say, over 20, a scatterplot can be very illuminating. Another program, called RANKPLOT (program 10-2), is designed to graph correlations when both X and Y are in the form of ranks.

PROGRAM 10-2. RANKPLOT. Rank-order correlation scatter chart.

```

10 'PROGRAM 10-2: "RANKPLOT" -- RANK-ORDER CORRELATION SCATTERPLOT
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR TRS-80 MODEL III AND DMP-400 PRINTER
20 CLEAR 1000
30 DIM YX$(50),CF(12)
50 LPRINT CHR$(30);CHR$(27);CHR$(31);          'BOLD PRINTING
55 GR$=CHR$(18)+CHR$(27)+CHR$(16)              'FOR POSITIONING
60 P0$=GR$+CHR$(0)+CHR$(0)                      'CARRIAGE RETURN W/O LINE FEED
70 BK$=CHR$(18)+STRING$(30,128)+CHR$(30)        'BLANK SPACING

```

```

80 INPUT "NAME OF X VARIABLE";VX$
85 INPUT "NAME OF Y VARIABLE";VY$
90 GOTO 120
100 QAX=FIX(A/256): AA%=FIX(A-256*QAX): RETURN 'CONVERSION SUBROUTINE
120 GOSUB 5000 'TOP X AXIS, LABELING, NUMBERING
150 GOSUB 2000: GOSUB 4000 'VERTICAL LINES & LINE FEED
160 N=1
180 READ TAG$,Y,X
190 IF TAG$="DONE" THEN GOTO 270
200 Y=Y+100: X=X+100
210 YX$(I)=STR$(Y)+TAG$+STR$(X) 'CONCATENATION OF X AND Y RANKS
220 D=Y-X: DSQ=D*D: CUMDSQ=CUMDSQ+DSQ 'SQUARE OF DIFF., CUMULATION OF SQUARES
230 PRINT "N=";N;" YX$(I)=";YX$(I);
240 I=I+1: N=N+1
260 GOTO 180
270 SPEARMAN=1-(6*CUMDSQ/(N^3-N)): PRINT "SPEARMAN r=";SPEARMAN 'CORR. COEFF.
280 ***** PRINTING OF DATAPOINTS (ZIP CODES, YEARS, ETC.) *****
290 FOR M=5 TO 50 STEP 5
300 GOSUB 2000: GOSUB 4000: GOSUB 2000: GOSUB 4000
310 FOR Q=1 TO 10: CF(Q)=0: NEXT Q 'SET CELL FREQ'S TO ZERO
312 Q=0
315 GOSUB 7000 'Y-AXIS NUMBERING
320 FOR I=0 TO N-1
330 PRINT M-4,M,YX$(I)
340 IF VAL(LEFT$(YX$(I),4))-100>(M-5) AND VAL(LEFT$(YX$(I),4))-100<M+1 THEN
GOSUB 6000: GOTO 340
350 NEXT I
360 FOR R=1 TO 4-NL: GOSUB 2000: GOSUB 4000: NEXT R 'VERTICAL LINES & LINE FE
EDS
370 NL=0
380 GOSUB 3000: GOSUB 2000: GOSUB 4000 'HORIZONTAL LINE, LINE FEED
390 NEXT M
400 GOSUB 4000: GOSUB 4000
410 LPRINT GR$;CHR$(0);CHR$(75);CHR$(30);"SPEARMAN r =" ;P0$;CHR$(0);CHR$(150);SP
EARMAN
520 END
2000 '***** VERTICAL LINES *****
2010 LPRINT GR$;CHR$(0);CHR$(60);CHR$(255);
2015 FOR Z=1 TO 10: LPRINT STRING$(59,128);CHR$(255);: NEXT Z
2050 RETURN
3000 '***** HORIZONTAL LINE *****
3010 LPRINT GR$;CHR$(0);CHR$(60);: FOR Z=1 TO 600: LPRINT CHR$(136);: NEXT Z
3020 RETURN
4000 '***** 6/72" LINE FEED ONLY *****
4010 FOR C=1 TO 6: LPRINT CHR$(27);CHR$(50);: NEXT C
4020 RETURN
5000 '***** TOP X AXIS, LABELING, NUMBERING *****
5003 LPRINT GR$;CHR$(0);CHR$(6);CHR$(30);"RANK IN Y (" ;VY$;")";";
5010 LPRINT GR$;CHR$(1);CHR$(45);CHR$(30);"RANK IN X (" ;VX$;")";
5015 GOSUB 4000: GOSUB 4000: GOSUB 4000
5017 LPRINT GR$;CHR$(0);CHR$(75);CHR$(30);"46-50";BK$;"41-45";BK$;"36-40";BK$;"3
1-35";BK$;"26-30";BK$;"21-25";BK$;"15-20";BK$;"11-15";BK$;CHR$(27);CHR$(6);
"6-10";BK$;CHR$(27);CHR$(6);" 1-5";CHR$(13);

```

```

5020 LPRINT GR$;CHR$(0);CHR$(30);CHR$(30);CHR$(27);CHR$(18);CHR$(167);CHR$(27);C
HR$(19);CHR$(8);CHR$(13);CHR$(20);CHR$(27);CHR$(30);CHR$(245);CHR$(27);CHR$
(28);CHR$(19);
5030 LPRINT P0$;GR$;CHR$(0);CHR$(60);CHR$(248);STRING$(250,136);STRING$(250,136)
;STRING$(99,136);CHR$(248);
5040 FOR A=120 TO 600 STEP 60: GOSUB 100: LPRINT GR$;CHR$(QA%);CHR$(AA%);CHR$(24
8);: NEXT A
5050 GOSUB 4000
5060 RETURN
6000 '***** PRINTING OF DATAPOINTS *****
6010 XX=VAL(RIGHT$(YX$(I),3))-100: Q=FIX(XX/5)+1: IF XX=50 THEN Q=10
6015 PRINT "XX=";XX,"Q=";Q
6020 IF CF(Q)=1 OR CF(Q)=3 OR CF(Q)=5 OR CF(Q)=7 OR CF(Q)=9 OR CF(Q)=11 THEN CP=
28 ELSE CP=0
6030 IF CF(Q)=2 OR CF(Q)=4 OR CF(Q)=6 OR CF(Q)=8 OR CF(Q)=10 OR CF(Q)=12 THEN NL
=NL+2: GOSUB 2000: GOSUB 4000: GOSUB 2000: GOSUB 4000
6040 A=670-60*Q+CP: GOSUB 100: LPRINT GR$;CHR$(QA%);CHR$(AA%);CHR$(30);CHR$(27);
CHR$(18);MID$(YX$(I),5,2);CHR$(27);CHR$(19);
6050 CF(Q)=CF(Q)+1
6060 I=I+1
6070 RETURN
7000 '***** Y-AXIS NUMBERING *****
7005 IF M=5 OR M=10 THEN GP$=CHR$(18)+STRING$(6,128)+CHR$(30) ELSE GP$=CHR$(30)
7008 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(19);
7010 LPRINT P0$;CHR$(30);GP$;CHR$(27);CHR$(18);M-4;"-";M;CHR$(27);CHR$(19);: RET
URN
10000 '***** DATA (STATE, PROPERTY CRIME, VIOLENT CRIME)
10010 DATA NV,1,6
10020 DATA AZ,2,15
10030 DATA CO,3,17
10040 DATA FL,4,3
10050 DATA CA,5,5
10060 DATA HI,6,39
.
.
.
10450 DATA KY,46,32
10460 DATA PA,47,29
10470 DATA ND,48,50
10480 DATA SD,49,49
10490 DATA WV,50,44
10500 DATA DONE,99,99

```

Figure 10-4 shows one of the printouts from RANKPLOT. In this example the rankings of all fifty states in two more general types of crime—property crime (X) and violent crime (Y)—are correlated. The fifty ranks for each type of crime are divided into groups of five to make it handy for locating the data points (states' U.S. Postal Service abbreviations) in a ten-by-ten grid. The RANKPLOT program also calculates the

RANK IN Y (VIOLENT CRIME RATE)		RANK IN X (PROPERTY CRIME RATE)									
		46-50	41-45	36-40	31-35	26-30	21-25	15-20	11-15	6-10	1-5
1											
1 - 5								AZ CO		CA	FL
6 - 10			HI			VA		OR	DE	NH	
11 - 15									TX NJ AK		NY MD
16 - 20			UT			CT RI			MA	MI	
21 - 25					KS		OK	GA		LA	SC
26 - 30	VT				WY		OH IL	MO			
31 - 35		VI MT MN	IN				AL				
36 - 40	IA				VA	TN	NC				
41 - 45	NH ME	NE	MS		AR						
46 - 50					KY	PA					

FIGURE 10-4. A rank-order correlation scatterplot (RANKPLOT program). U.S. Postal Service abbreviations are used to indicate each state's standings in both crimes of violence and property crimes.

Spearman correlation coefficient; in this case the r_s turns out to be $+ .70$ on the scale from -1 to $+1$.

As in the PEARSONR program, X and Y measures are concatenated into a string array called YX\$(I), consisting of a Y rank on the left end, its paired X rank on the right end, and the state's abbreviation in the middle (line 210). But unlike the PEARSONR program, RANKPLOT uses a search rather than sort routine. In line 290 (FOR M=5 TO 50 STEP 5) the search begins, with M representing the intervals (1-6, 6-10, . . . , 46-50) of the Y ranks. At each value of M, the program scans the whole list of YX\$(I) strings, finds the rank values of Y(I) that fall within M's interval, and prints the state abbreviation that goes with each Y(I) value that is found. The value of X(I) that is stored in the same YX\$(I) value with these Y(I) and state-abbreviation items determines the location (cell in the row) in which the printing is done.

There's a special feature in the program that requires an-

a situation in which the different pairs don't come from different subjects (states, people, etc.). The typical situation is one in which you have two sets of ranks for a single subject (person, team, company, etc.) for each year over many years.

In figure 10-5 we can see how a major-league baseball team's position in the final standings from 1901 to 1983 has correlated with its rank in team batting average. This time four-digit years serve as data points and thereby provide a lot of extra information that geometric symbols wouldn't (e.g., in which years did the team finish first in the standings without being the best team in batting?). A ten-by-ten grid is used again, but the numbering of the rows and columns is in whole numbers from 1 to 10 instead of intervals from 1-5 to 46-50.

When we have a lot of scatterplots like the one in figure 10-5, we can see at a glance how close each one is to having all of its year prints lining up along a diagonal row of cells (from one corner of the grid to its opposite corner), as opposed to a random pattern of data points. We can also compare their Spearman r_s values to get a more precise estimate of the strengths of correlations.

The main changes in RANKPLOT to fit this baseball example were, besides the renumbering of the axes, narrowing the all-important search statement (line 340) from a five-rank range to a single rank at a time, and expanding the size of the scatterplot to allow room for up to a dozen four-digit prints in each cell. The enlargement requires a fifteen-inch printer, but although the figure on crime rates was printed by RANKPLOT on one of that size, that program could easily be revised to fit on eight-inch paper.

Ladder Graphs

A ladder graph is not a common type of graph with a standard name. In fact, other names, such as linear graph, thermometer graph, and even totem-pole graph, have been proposed for this graphic form. Whatever you want to call them, ladder graphs have a vertical scale along which the names of items (people, companies, countries, etc.) are located according to their value on the scale.

In figure 10-6 we have an example of a ladder graph. Program 10-3, called TOTMPOLE, produced it.

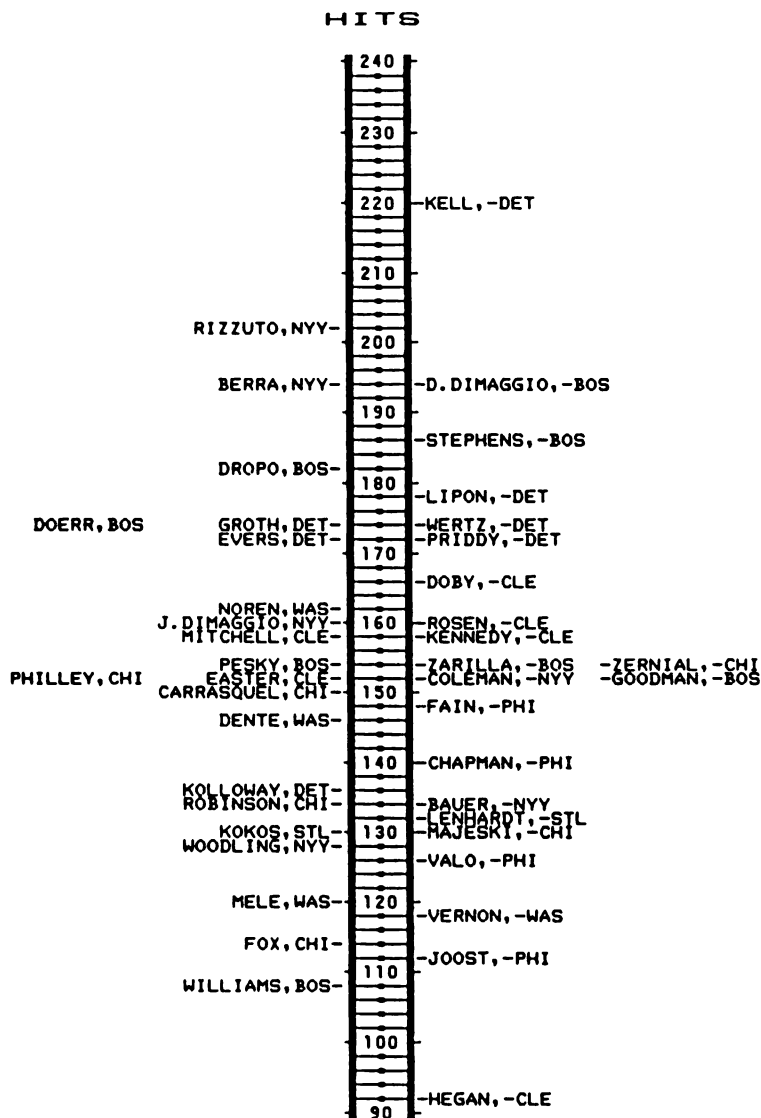


FIGURE 10-6. A ladder graph produced by the TOTMPOLE program for the DMP-400.

PROGRAM 10-3. TOTMPOLE. Ladder graph.

```

10 'PROGRAM 10-3: "TOTMPOLE" -- LADDER GRAPH
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 MODEL III OR 4, TRS-80 PRINTERS
30 DIM A$(100)
40 LPRINT CHR$(30);CHR$(27);CHR$(19);
                                     'STANDARD DENSITY
50 LFT=4: CNTR=6: RGT=10
60 V=7: STAT$="HITS": HI=240: LO=60: STEPSIZE=-2
70 OPEN "I",1,"AL50BATG"
                                     'OPEN DATA FILE (1950 AMERICAN LEAGUE BATTING)
80 INPUT #1, YEAR$, PLYR$, TEAM$, AB, R, H, HR, RBI, BB, SB, BA, SA

```

```

90 IF YEAR$="-1" THEN GOTO 80
100 IF EOF(1) THEN CLOSE 1: PRINT "FILE CLOSED": GOTO 160
110 N=N+1
120 PRINT N;" ";YEAR$;" ";PLYR$;" ";TEAM$;" ";AB;R;H;HR;RBI;BB;SB;BA;SA
130 C=100: A$(N)=STR$(H+C)+"-"+TEAM$+"-"+PLYR$: USG$="###" 'FILL STRING ARRAY
140 GOTO 80 ' (FOR SORTING)
150 REM -- LINES 150-210 PERFORM BUBBLE SORT OF PLAYERS ON THE STATISTIC (HITS)
160 PRINT "NOW SORTING THE PLAYERS BY ";STAT$
170 SWITCH=0
180 FOR I=0 TO N
190 IF A$(I)<A$(I+1) THEN SWAP A$(I),A$(I+1): SWITCH=SWITCH+1
200 NEXT I
210 P=P+1: PRINT "PASS=";P
220 IF SWITCH>0 THEN GOTO 170 ELSE PRINT TAB(0);"SORTED"
230 LPRINT CHR$(13): LPRINT CHR$(30);STAT$; 'PRINT LADDER HEADING
240 LPRINT CHR$(13)
250 FOR I=0 TO N-1 'LINES 250-290 DISPLAY SORTED ARRAY ON SCREEN & PRINTER
260 PRINT A$(I)
270 SCORE=VAL(LEFT$(A$(I),LFT))-C
280 LPRINT CHR$(30);MID$(A$(I),RGT);,;MID$(A$(I),CNTR,3);" ";SCORE;" ";
290 NEXT I
300 LPRINT CHR$(13)
310 BS$=CHR$(30)+CHR$(8)+CHR$(4)+CHR$(18) 'BACKSPACE (2 DOTS)
320 BN$=CHR$(30)+CHR$(8)+CHR$(1) 'BACKSPACE (1/2 DOT)
330 PN$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(10) '266th DOT-COLUMN POSITION
340 PX$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(254) '254th " " "
350 TC=269-6*LEN(STAT$): T1%=FIX(TC/256): T2%=FIX(TC-256*T1%)
360 LPRINT CHR$(27);CHR$(16);CHR$(T1%);CHR$(T2%);CHR$(30);CHR$(27);CHR$(14);STAT$;CHR$(27);CHR$(15);CHR$(13);CHR$(13);
370 FOR I=0 TO N-1: SCORE=VAL(LEFT$(A$(I),LFT))-C: IF SCORE>HI OR SCORE<LO THEN LPRINT MID$(A$(I),RGT);", ";MID$(A$(I),CNTR,3);USING USG$;SCORE;: LPRINT CHR$(13);
380 IF SCORE>HI OR SCORE<LO THEN LPRINT MID$(A$(I),RGT);", ";MID$(A$(I),CNTR,3);USING USG$;SCORE;: LPRINT CHR$(13);
390 NEXT I
400 I=0: FOR J=HI TO LO STEP STEPSIZE
410 IF VAL(LEFT$(A$(I),LFT))-C>HI THEN I=I+1: GOTO 410
420 U=0
430 PRINT J, A$(I),J+(STEPSIZE)
440 IF SCORE<=J AND SCORE>=J+(STEPSIZE) THEN U=U+1: PRINT "SCORE=";SCORE,J: GOSUB 540: GOTO 440
450 IF J<100 THEN SP=5 ELSE SP=2
460 J$=STR$(J): IF RIGHT$(J$,1)="0" THEN LPRINT CHR$(18);PX$;STRING$(SP,128);BN$;CHR$(30);CHR$(27);CHR$(18);INT(J);CHR$(27);CHR$(19);: LPRINT CHR$(18);PX$;BS$;STRING$(6,136);STRING$(26,128);STRING$(6,136);
470 IF RIGHT$(J$,1)="0" THEN DTC=128: CCD=128: ELSE DTC=136: CCD=156
480 LPRINT PX$;CHR$(18);STRING$(4,255);STRING$(11,DTC);STRING$(3,CCD);STRING$(11,DTC);STRING$(4,255);
490 FOR LF=1 TO 6: LPRINT CHR$(27);CHR$(50);: NEXT LF
500 LPRINT CHR$(18);PX$;STRING$(4,131);STRING$(25,128);STRING$(4,131);CHR$(27);CHR$(50);CHR$(27);CHR$(50);
510 NEXT J
520 PRINT "JOB DONE"
530 STOP

```

```

540 '***** PRINTING OF PLAYER AND TEAM *****
550 IF U<3 THEN RP=32: LP=252: CL=45
560 IF U=3 OR U=4 THEN RP=122: LP=162: CL=32
570 IF U>4 THEN RP=212: LP=72: CL=32
580 I$=STR$(I)
590 IF RIGHT$(I$,1)="0" OR RIGHT$(I$,1)="2" OR RIGHT$(I$,1)="4" OR RIGHT$(I$,1)=
   "6" OR RIGHT$(I$,1)="8" THEN LPRINT CHR$(27);CHR$(16);CHR$(1);CHR$(RP);CHR$(
   30);MID$(A$(I),RGT-1);", ";MID$(A$(I),CNTR-1,4);: I=I+1: GOTO 610
600 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(LP-6*LEN(MID$(A$(I),CNTR-1)));CHR$(30)
   ;MID$(A$(I),RGT);", ";MID$(A$(I),CNTR,3);CHR$(CL);: I=I+1
610 RETURN

```

In producing the printout, the TOTMPOLE program places the names of baseball players on both sides of a scale. The scale represents the number of hits by players over a whole season (1950), and all American League players in 1950 having four hundred or more official times at bat have their names and team abbreviations in the graph. In line 80 the program inputs these names and the players' hit totals (along with much additional, unused information about other batting statistics) from a datafile. For the total hits measure (H), an array called A\$(I) stores the pertinent data. The players are then bubble-sorted (lines 150–220) so that they can be listed and read in order of their hit totals.

After the sorting, TOTMPOLE lists the reordered array (line 250–290) and starts drawing the scale (at line 370) from top to bottom so that the length of the scale runs perpendicular to the printhead's movement. As the program steps down to each new value (hit total) on the scale, it checks to see if that value is connected to any player's name. When a matching value turns up, the name is printed to the left or right of the scale. The printer's built-in characters are used both for printing the names and for numbering the scale.

The first matching name is printed on the left, the next on the right, and so forth. Using both sides of the scale this way allows room for ties, but if a three-, four-, five-, or six-way tie occurs, the names are printed progressively farther from the scale so as not to be on top of each other. Printers with eight-inch paper will probably have to use condensed (16.7-CPI) characters to fit in the names when five- or six-way ties occur. On the other hand, if you want to keep the whole graph sleek and trim, change the program so that the third to sixth names in a tie will be stored and printed in a group below the graph,

using the same print density as with the first and second names, and then cut and paste the names manually in locations on the scale as space permits.

If such a manual method seems too much like admitting defeat as a programmer, you could change the program so that the third and fourth names in a tie are printed directly above the first and second names, using reverse line feeds and after checking whether there is enough space left for this by names that have already been printed before the tie occurred. With forward line feeds, you can return the printhead to where it was before the reverse feeds and then temporarily drop down to the positions below the first and second names, in order to print the fifth and sixth names in the tie. Again, a check for space would be needed, but these space checks are made fairly easy to program by the fact that the names and their numerical values have been stored in an ordered string array. If space is not available for a given name-print, you still have the other two methods to fall back on, and those options could be built into the programming. Incidentally, only small changes in the TOTMPOLE program are needed to make it print all the names on the right (or left) of the scale.

For another illustration of a ladder graph, see the final chapter (figure 12-4).

Pie Charts

If there is any kind of graph that usually comes out better using a screen dump instead of direct programming of the printer, it is the pie chart (also known as the circle graph). Programming the drawing of a pie chart on the computer's screen is not too difficult, and the labeling of the chart can also be done by part of the same program. Both matters can be handled by several different pie-chart programs that have appeared in microcomputer graphics books. And if the screen has the resolution that we find on the IBM PC, a fairly good quality can be achieved in the hard-copy version of the chart when it is dumped to a printer (or plotter). Do-it-yourself IBM users are advised to use Baenziger's graphic screen dump (GRUMP) program (*PC World*, December, 1983, pp. 176–191) in combination with the pie-chart screen-drawing program from the books or an IBM PC demonstration disk.

For direct programming of pie charts on a printer, consider the PIECHART/EPS program (program 10-4). This too can be viewed as a combination of two programs, the screen-drawing program for an exploded pie chart presented by Hearn and Baker (*Microcomputer Graphics*, Prentice-Hall, 1983, pp. 245-47) and our printer-drawing program called POLARRAY (program 5-6).

PROGRAM 10-4. PIECHART/EPS. Exploded-slice pie-chart program for bit-image drawing (for IBM-Epson printers).

```

10 'PROGRAM 10-4: "PIECHART/EPS" -- EXPLODED PIE CHART FOR IBM-EPSON PRINTERS
20 'EXTENSION OF PROGRAM 12-1 IN D.HEARN & M.P. BAKER, 'MICROCOMPUTER GRAPHICS'
    BOOK, PRENTICE-HALL, 1983
30 'FOR THE EPSON FX-80 PRINTER AND TRS-80 MODEL III (USE BOOTHE'S DRIVER)
40 LPRINT CHR$(27)"@";
50 CLEAR 4000
60 DEFINT X,Y
70 DIM XY$(500),Y(500)
80 GOTO 190
90 '***** PRINTING OF DRAWING DOT *****
100 Y(I)=.9*Y(I): G2%=FIX(Y(I)/256): G1%=FIX(Y(I)-256*G2%)
110 PRINT "G1=";G1%;"G2=";G2%
120 IF G2%=0 THEN S1=0: S2=0: S3=0
130 IF G2%=1 THEN S1=255: S2=0: S3=0
140 IF G2%=2 THEN S1=255: S2=255: S3=0
150 IF G2%=3 THEN S1=255: S2=255: S3=255
160 LPRINT CHR$(27)"Z"CHR$(G1%)CHR$(G2%);STRING$(S1,0);STRING$(S2,0);STRING$(S3,
    0);STRING$(G2%,0);STRING$(G1%,0);DP$;CR$;
170 RETURN
180 '-----
190 INPUT "NO. OF SLICES (2 TO 8)";N
200 INPUT "EXPLODE WHICH SLICE (1 TO 8)";E
210 R=250
220 XN=1200: YN=330
230 YA=.6
240 FOR K=1 TO N
250   READ N$(K),V(K)
260   IF N$="END" THEN 290
270   T=T+V(K)
280 NEXT K
290 CLS: B=0: S=0
300 RE=360*3.14159/180
310 FOR K=1 TO N
320   XC=XN: YC=YN
330   S=S+V(K)
340   A=RE*S/
350   IF K<>E THEN 460
360   AC=B+(A-B)/2
370   XE=XC+R/5*COS(AC)
380   YE=YC+R/5*SIN(AC)*YA

```

'RADIAN EQUIV.
'NORMAL CENTER
'CUMUL. SECTION VALUE
'ANGLE IN RADIAN
'EXPLODE?
'HALF-ANGLE

```

390  XP=XE+R*COS(B)
400  YP=YE+R*SIN(B)*YA
410  GOSUB 780
420  XP=XC+R*COS(A)
430  YP=YC+R*SIN(A)*YA
440  GOSUB 590
450  XC=XE: YC=YE
460  FOR A1=B TO A STEP 10/R
470      XP=XC+R*COS(A1)
480      YP=YC+R*SIN(A1)*YA
490      XY$(I)=STR$(INT(XP+300))+ "A" +STR$(INT(YP)): I=I+1
500  NEXT A1
510  GOSUB 590
520  B=A
530  NV=I
540  NEXT K
550  GOSUB 840                      'SORT OF XY$(I) VALUES
560  GOSUB 1140
570  END
580  '-----
590  '***** X-Y COORDINATES OF SECTION LINES *****
600  IF XC<>XP THEN 670
610  IF YC>YP THEN ST=10 ELSE ST=-10
620  FOR Y=YC TO YP STEP ST: XY$(I)=STR$(XC+300)+ "A" +STR$(INT(Y))
630  I=I+1
640  NEXT Y
650  RETURN
660  '-----
670  M=(YP-YC)/(XP-XC)
680  IF ABS(M)>1 THEN ST=ABS(10/M) ELSE ST=10
690  IF XC>XP THEN ST=-ST
700  FOR X=XC TO XP STEP ST
710      Y=M*(X-XC)+YC
720      XY$(I)=STR$(INT(X+300))+ "A" +STR$(INT(Y))
730  LPRINT I;
740      I=I+1
750  NEXT X
760  RETURN
770  '-----
780  '***** X-Y COORDS OF EXPLODED-SECTION LINES *****
790  XC=XE: YC=YE
800  GOSUB 590
810  XC=XN: YC=YN
820  RETURN
830  '-----
840  '***** SHELL-METZNER SORT OF XY$(I) *****
850  SOUND 6,5: SOUND 4,10: SOUND 2,8
860  PRINT "SORT"
870  P=0
880  Q=NV
890  Q=INT(Q/2)
900  IF Q=0 THEN 1090
910  P=P+1

```

```

920 PRINT P
930 FOR BL=0 TO Q-1
940 I=BL
950 J=BL+Q
960 SW=0
970 IF XY$(I)<=XY$(J) THEN 1020
980 SW=1
990 B$=XY$(I)
1000 XY$(I)=XY$(J)
1010 XY$(J)=B$
1020 I=J
1030 J=J+Q
1040 IF J<NV THEN GOTO 970
1050 IF SW=0 THEN GOTO 1070
1060 GOTO 940
1070 NEXT BL
1080 GOTO 890
1090 PRINT "SORT DONE": SOUND 4,5: SOUND 5,8
1100 GOTO 1120
1110 FOR I=0 TO NV-1: LPRINT XY$(I);: NEXT I
1120 RETURN
1130 '-----
1140 '***** PRINTOUT ON EPSON FX-80 *****
1150 LPRINT CHR$(27)"@" 'MASTER RESET
1160 LPRINT CHR$(27)"3"CHR$(4) 'SET 4/216" LINE FEED
1170 DP$=CHR$(27)+"Z"+CHR$(8)+CHR$(0)+STRING$(8,28) 'QUADRUPLE-DENSITY DOT
1180 CR$=CHR$(13)+CHR$(27)+"j"+CHR$(4) 'CARR. RET. W/ REVERSE LINE FEED
1190 KL=1150: KH=1160
1200 FOR I=0 TO NV-1
1210 PRINT I,KL,KH
1220 IF VAL(LEFT$(XY$(I),5))>KH THEN LPRINT CHR$(10);: KL=KH: KH=KH+5: GOTO 12
1230 IF VAL(LEFT$(XY$(I),5))>KL AND VAL(LEFT$(XY$(I),5))<KH THEN PRINT XY$(I)
1240 NEXT I
1250 RETURN
1260 '-----
1270 DATA A,20,B,24,C,18,D,10,E,17,F,22,END

```

PROGRAM 10-5. PIECHART/TRS. Exploded-slice pie-chart program for TRS-80 printers.

```

10 'PROGRAM 10-5: "PIECHART/TRS" -- EXPLODED PIE CHART (TRS-80 MOD. 4 VERSION)
15 'EXTENSION OF PROGRAM 12-1 IN D.HEARN & M.P.BAKER, 'MICROCOMPUTER GRAPHICS' B
    OOK, PRENTICE-HALL, 1983
20 LPRINT CHR$(30);CHR$(27);CHR$(20); 'CONDENSED PRINTING DENSITY
30 CLEAR 4000
40 DEFINT X,Y
50 DIM XY$(500),Y(500)
60 GOTO 140
69 '-----
70 Y(I)=.375*Y(I): G1%=FIX(Y(I)/256): G2%=FIX(Y(I)-256*G1%)

```



```

80 PRINT "G1=";G1%;"G2=";G2%
90 IF G2%=9 THEN G2%=8: AS%=CHR$(30)+CHR$(27)+CHR$(2)+CHR$(18): ELSE AS%=""
100 '(LINE 42 IS A DETOUR AROUND TROUBLE-CODE 9, DROPPING IT TO 8 AND
110 'ADDING A SINGLE-DOT SPACE WITH AS% BEFORE THE DATAPoint PRINT.)
120 LPRINT CHR$(27);CHR$(16);CHR$(G1%);CHR$(G2%);AS%;DP%;
130 RETURN
139 '-----
140 INPUT "NO. OF SLICES (2 TO 8)";N
150 INPUT "EXPLODE WHICH SLICE (1 TO 8)";E
160 R=250
170 XN=1200: YN=330
180 YA=.5
190 FOR K=1 TO N
200   READ N$(K),V(K)
210 IF N$="END" THEN 240
220   T=T+V(K)
230 NEXT K
240 CLS: B=0: S=0
250 RE=360*3.14159/180 'RADIAN EQUIV.
260 FOR K=1 TO N
270   XC=XN: YC=YN 'NORMAL CENTER
280   S=S+V(K) 'CUMUL. SECTION VALUE
290   A=RE*S/T 'ANGLE IN RADIAN
300   IF K<>E THEN 410 'EXPLODE?
310   AC=B+(A-B)/2 'HALF-ANGLE
320   XE=XC+R/5*COS(AC)
330   YE=YC+R/5*SIN(AC)*YA
340   XP=XE+R*COS(B)
350   YP=YE+R*SIN(B)*YA
360   GOSUB 690
370   XP=XC+R*COS(A)
380   YP=YC+R*SIN(A)*YA
390   GOSUB 530
400   XC=XE: YC=YE
410   FOR A1=B TO A STEP 10/R
420     XP=XC+R*COS(A1)
430     YP=YC+R*SIN(A1)*YA
440     XY$(I)=STR$(INT(XP+300))+ "A"+STR$(INT(YP)): I=I+1
450   NEXT A1
460   GOSUB 530
470   B=A
480 NV=I
490 NEXT K
500 GOSUB 740 'SORT OF XY$(I) VALUES
510 GOSUB 1030
520 END
528 '-----
530 'X-Y COORDINATES OF SECTION LINES
540 IF XC<>XP THEN 600
550 IF YC>YP THEN ST=10 ELSE ST=-10
560 FOR Y=YC TO YP STEP ST: XY$(I)=STR$(XC+300)+ "A"+STR$(INT(Y))
570 I=I+1
580 NEXT Y
590 RETURN

```

```

598 '-----
600 M=(YP-YC)/(XP-XC)
610 IF ABS(M)>1 THEN ST=ABS(10/M) ELSE ST=10
620 IF XC>XP THEN ST=-ST
630 FOR X=XC TO XP STEP ST
640   Y=M*(X-XC)+YC
650   XY$(I)=STR$(INT(X+300))+ "A" +STR$(INT(Y))
660   I=I+1
670 NEXT X
680 RETURN
688 '-----
690 'X-Y COORDS OF EXPLODED-SECTION LINES
700 XC=XE: YC=YE
710 GOSUB 530
720 XC=XN: YC=YN
730 RETURN
738 '-----
740 'SHELL-METZNER SORT OF XY$(I)
750 SOUND 6,5: SOUND 4,10: SOUND 2,8
760 PRINT "SORT"
770 P=0
780 Q=NV
790 Q=INT(Q/2)
800 IF Q=0 THEN 990
810 P=P+1
820 PRINT P
830 FOR BL=0 TO Q-1
840   I=BL
850   J=BL+Q
860   SW=0
870   IF XY$(I)<=XY$(J) THEN 920
880   SW=1
890   B$=XY$(I)
900   XY$(I)=XY$(J)
910   XY$(J)=B$
920   I=J
930   J=J+Q
940   IF J<NV THEN GOTO 870
950   IF SW=0 THEN GOTO 970
960   GOTO 840
970 NEXT BL
980 GOTO 790
990 PRINT "SORT DONE": SOUND 4,5: SOUND 5,8
1000 GOTO 1020
1010 FOR I=0 TO NV-1: LPRINT XY$(I);: NEXT I
1020 RETURN
1028 '-----
1030 'PRINTOUT ON TRS-80 DMP-400 PRINTER
1040 LF$=CHR$(27)+CHR$(50)+CHR$(27)+CHR$(51)
1050 DP$=CHR$(18)+STRING$(4,156)
1060 KL=1150: KH=1160
1070 FOR I=0 TO NV-1
1080   PRINT I,KL,KH
1090   IF VAL(LEFT$(XY$(I),5))>KH THEN LPRINT LF$;: KL=KH: KH=KH+5: GOTO 1080

```

```

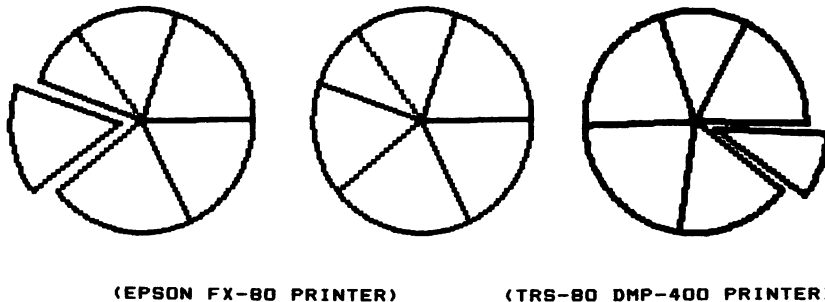
1100 IF VAL(LEFT$(XY$(I),5))>KL AND VAL(LEFT$(XY$(I),5))=<KH THEN PRINT XY$(I)
      : Y(I)=800-VAL(RIGHT$(XY$(I),3)): GOSUB 70
1110 NEXT I
1120 RETURN
1128 '-----
1130 DATA A,20,B,12,C,18,D,10,E,17,F,22,END

```

Printouts from PIECHART/EPS, written for IBM-Epson printers, and from PIECHART/TRS (program 10-5) are shown in figure 10-7. The left and center examples, printed by the Epson FX-80, show that the same program can be used for both exploded and unexploded pie charts. To the right of these is an exploded pie chart printed by the DMP-400 using the TRS-80 version of the PIECHART program.

At this point a major shortcoming becomes evident—unlike most screen dumps, these PIECHART programs don't provide any labeling of the pie sections. There doesn't seem to be any simple way to do this labeling on the printer, although there is the possibility that after the pie chart itself is drawn, the X-Y coordinates used in its drawing can also be used to position labels, which are printed with built-in characters. Even without labeling, direct-programmed pie charts may be very useful because of their high resolution, and you can nearly always get a better finished product by labeling them manually. Furthermore, in illustrations involving many pies in a group (see an example of this in chapter 12) the usual labels would only get in the way.

FIGURE 10-7. Printouts from the PIECHART programs for the Epson FX-80 (left and center) and the DMP-400. (Adapted by permission of Prentice-Hall, Inc., from *Microcomputer Graphics: Techniques and Applications* by Donald Hearn and M. Pauline Baker.)



Still, it's an unpleasant surprise to discover a labeling problem this difficult in printer graphics. On this somewhat sour note, we move from graphs to diagrams and tables, hoping for better luck (or skill).

Diagrams, Maps, Tables, and Graphics

THERE are lots of graphic creations that aren't called graphs because they don't have the main purpose of presenting numerical data. The clearest examples are electrical circuit diagrams, organization charts, flow sheets, architectural drawings, and maps. When it comes to programming these for a dot-matrix printer, we often have a choice between the bit-image approach and using the printer's built-in block graphics.

Diagrams with Block and Line-Graphic Characters

Some diagrams—not very complex ones—can be drawn by printers without ever entering the graphic mode. Many printers, including all the recent TRS-80s, the IBM Matrix and Graphic printers, and the NEC 8023A have ready-made graphic characters that can be used in the ordinary text mode. The IBM and NEC characters are shown in figure 11-1, and the TRS-80 characters are shown in figure 11-2 (left side).

With a little study, you can see that most of these characters are designed to be fitted together to make diagrams of many types. They can also be used for bar graphs, but are probably useless for other types of graphs or for maps. Actually, it isn't very easy to find or dream up illustrations of their use even in diagrams—few have appeared in the micro publications and almost none in manuals for printers having these built-in characters. It is even harder to come up with examples in which all the drawing is done with them, as opposed to being mixed with bit-image drawing.

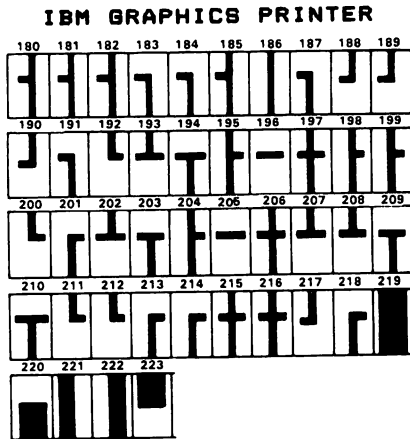
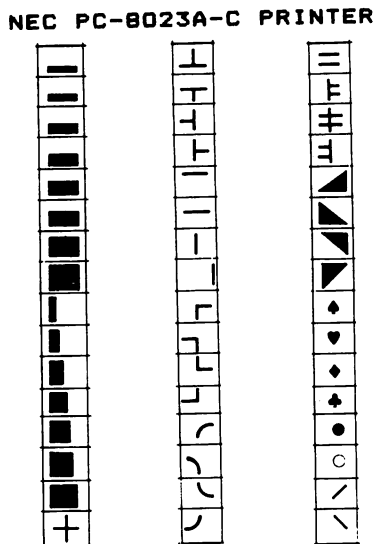


FIGURE 11-1. Raw materials for drawing diagrams in text mode: built-in graphic characters available on the IBM Graphics and NEC 8023A printers.



The NEC and IBM graphic symbols require $\frac{8}{72}$ -inch line feeds for fitting the characters together vertically, while the DMP-400's six-by-six symbols call for $\frac{6}{72}$ -inch spacing. Horizontal fitting is usually assured by the standard six-dot movement of the printhead after printing any text-mode character.

Printers that don't have these built-in graphic characters can easily make them out of dots in the graphic mode. As with homemade alphabetic characters, block and line symbols that are identical to Radio Shack's six-by-six ready-mades can be defined in strings and referred to later in a program by the string names. Better yet, you can make seven-

BUILT-IN 6 X 6BIT-IMAGE 7 X 7

CHR(225) :	■	■	■	STRING(3,128);STRING(4,143)
CHR(226) :	•	■	■	STRING(4,143);STRING(1,128)
CHR(227) :	■	■	■	STRING(4,248);STRING(3,128)
CHR(228) :	■	■	■	STRING(3,128);STRING(4,248)
CHR(229) :	◐	■	◐	STRING(4,143);STRING(4,248)
CHR(230) :	◑	■	◑	STRING(4,248);STRING(4,143)
CHR(231) :	■	■	■	STRING(7,143)
CHR(232) :	■	■	■	STRING(7,248)
CHR(233) :	■	■	■	STRING(4,255);STRING(3,128)
CHR(234) :	■	■	■	STRING(3,128);STRING(4,255)
CHR(235) :	◐	■	◐	STRING(4,255);STRING(3,143)
CHR(236) :	◑	■	◑	STRING(3,143);STRING(4,255)
CHR(237) :	■	■	■	STRING(4,255);STRING(3,248)
CHR(238) :	■	■	■	STRING(3,248);STRING(4,255)
CHR(239) :	■	■	■	STRING(7,255)
CHR(240) :	◐	■	◐	STRING(3,128);CHR(248);CHR(136)
CHR(241) :	◑	■	◑	STRING(7,136)
CHR(242) :	◐	■	◐	STRING(3,136);CHR(248);STRING(3,128)
CHR(243) :	◑	■	◑	STRING(3,136);CHR(248);STRING(3,136)
CHR(244) :	◐	■	◐	STRING(3,128);CHR(255);CHR(3,136)
CHR(245) :	◑	■	◑	STRING(3,128);CHR(255);CHR(3,128)
CHR(246) :	◐	■	◐	STRING(3,128);CHR(143);STRING(3,136)
CHR(247) :	◑	■	◑	STRING(3,136);CHR(143);STRING(3,128)
CHR(248) :	◐	■	◐	STRING(3,136);CHR(143);STRING(3,136)
CHR(249) :	◑	■	◑	STRING(3,136);CHR(255);STRING(3,128)
CHR(250) :	◐	■	◐	STRING(3,136);CHR(255);STRING(3,136)
CHR(251) :	◑	■	◑	CHR(255);CHR(191);CHR(159);CHR(143);CHR(135);CHR(131);CHR(129)
CHR(252) :	◐	■	◐	CHR(192);CHR(224);CHR(248);CHR(248);CHR(252);CHR(254);CHR(255)
CHR(253) :	◑	■	◑	CHR(129);CHR(131);CHR(135);CHR(143);CHR(159);CHR(191);CHR(255)
CHR(254) :	◐	■	◐	CHR(255);CHR(254);CHR(252);CHR(248);CHR(248);CHR(224);CHR(129)
(NO CHARACTER)	/			CHR(168);CHR(144);CHR(136);CHR(132);CHR(138);CHR(129)
(NO CHARACTER)	\			CHR(129);CHR(138);CHR(132);CHR(132);CHR(136);CHR(144);CHR(168)
(NO CHARACTER)	x			CHR(161);CHR(146);CHR(148);CHR(148);CHR(146);CHR(161)
(NO CHARACTER)				CHR(255);STRING(6,128)
(NO CHARACTER)				STRING(6,128);CHR(255)

FIGURE 11-2. Homemade seven-by-seven graphic characters using bit-image programming (right), compared with the DMP-400's built-in characters (left).

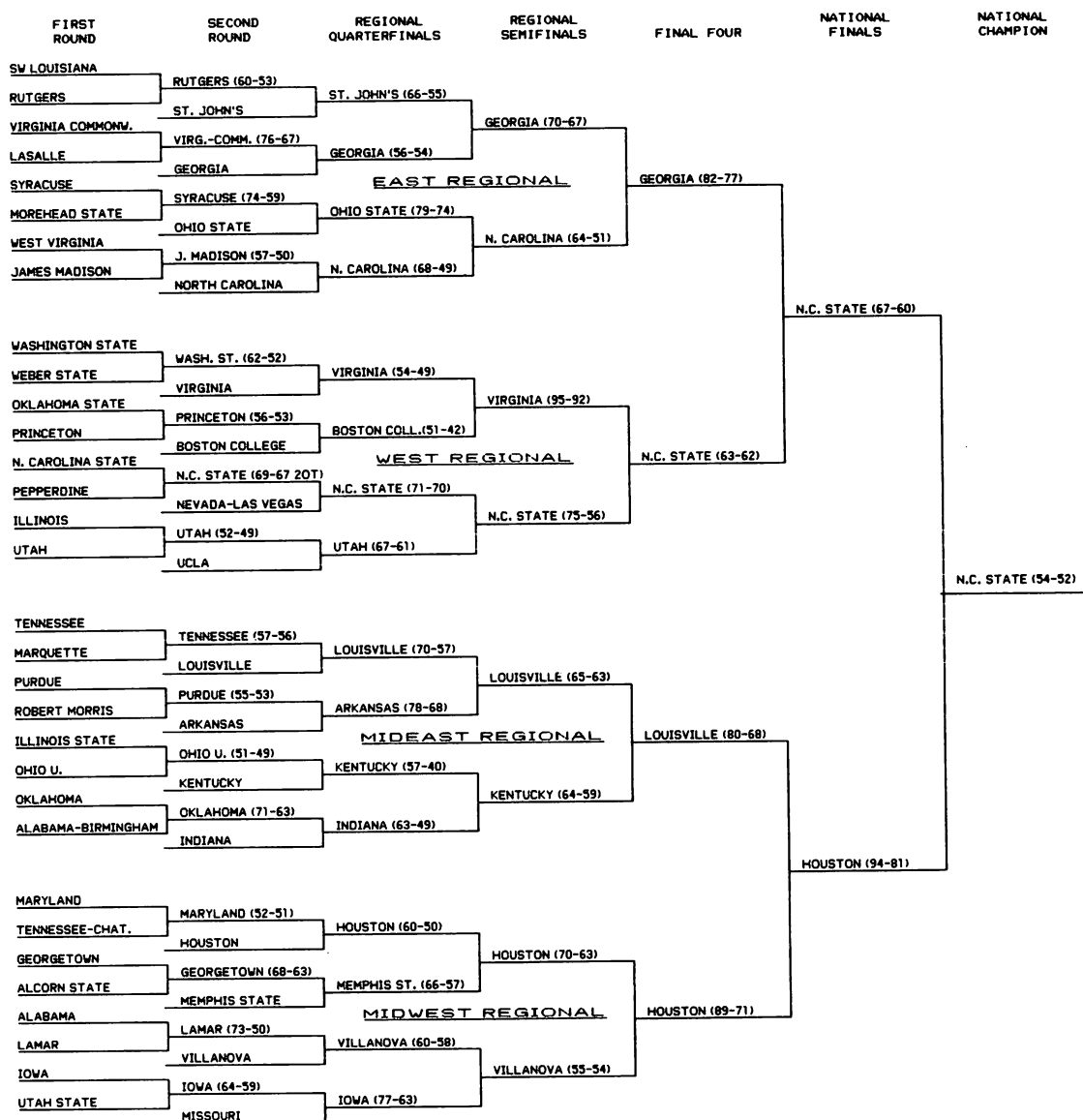
by-seven characters that are similar to these and gain the advantage of an odd number of dots—the middle dot of an odd number gives you a true center. That is, with eight-dot-high symbols the nearest you can get to the vertical center point is the fourth or fifth dot, and with six-dot-high symbols, the third or fourth dot. This can cause problems that you would avoid with seven-by-seven characters when it comes to fitting the symbols together in a diagram.

In addition to the DMP-400's six-by-six built-in characters, figure 11-2 shows a seven-by-seven bit-image version of each character and a few other symbols as well. You can see that it doesn't take very much typing to define each of the bit-image characters, and in the case of line symbols the seven-by-sevens are more perfectly symmetrical or better centered than the six-by-sixes.

A line-graphic diagram. For an example of the line-graphic characters in use, we have the tournament diagram in figure 11-3. Program 11-1 (TRNYGRAM) shows what it takes to draw the annual NCAA college basketball tourney.

Throughout the TRNYGRAM program the only graphic characters used are the DMP-400's straight horizontals and verticals, CHR\$(241) and CHR\$(245); two of the corners,

FIGURE 11-3. Using built-in line-graphic characters: a tournament diagram (TRNYGRAM program).



CHR\$(242) and CHR\$(247); and a reclining T-shaped character, CHR\$(244). Subroutine 2000 does all the drawing, and is reused for each of the four regions of the country. This requires that all East teams (eight first-round, eight winners of that round, four quarterfinalists, two semifinalists, and one of the Final Four teams) be listed in the DATA statements in that order. The same order has to be followed for the successive readings of the teams in the West, Mideast, and Midwest regions.

The more difficult part of the programming is interconnecting these regions (two at a time) to display the winners of the Final Four matches and the emergence of one of these national finalists as the national champion. That is the job of subroutines 3000–6000, and it involves several FOR-NEXT statements.

PROGRAM 11-1. TRNYGRAM. Tournament diagram using built-in line-graphic characters.

```

10 'PROGRAM 11-1: "TRNYGRAM" -- TOURNAMENT DIAGRAM WITH LINE-GRAPHIC CHARACTERS
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR TRS-80 COMPUTERS AND WIDEBODY PRINTERS
20 CLEAR 2000
30 PF$=CHR$(27)+CHR$(17)                                'PROPORTIONAL FONT
40 NM$=CHR$(27)+CHR$(19)                                'NORMAL (10-CPI) DENSITY
50 HL$=CHR$(27)+CHR$(32)+CHR$(27)+CHR$(14)              'ELONGATED ON
60 HE$=CHR$(27)+CHR$(15)+CHR$(27)+CHR$(31)              'ELONGATED OFF
70 SP$=CHR$(224)                                          'BLANK SPACE
80 LF$=CHR$(20)+CHR$(27)+CHR$(28)+CHR$(19)              '1/12" LINE FEED
90 '***** POSITIONING STRINGS *****
100 P0$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)+SP$
110 P1$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(108)
120 P2$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(216)
130 P3$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(68)
140 P4$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(176)
150 P5$=CHR$(27)+CHR$(16)+CHR$(2)+CHR$(28)
160 P6$=CHR$(27)+CHR$(16)+CHR$(2)+CHR$(136)
170 LPRINT CHR$(30);NM$;                                'RESET
180 '***** DATA INPUT *****
190 DIM R1$(32),W1$(16),R2$(16),QF$(16),SF$(8),REG$(4),FF$(4),NC$(2)
200 READ REG$(1): FOR N=1 TO 8: READ R1$(N): NEXT N 'ROUND 1
210 READ REG$(2): FOR N=9 TO 16: READ R1$(N): NEXT N
220 READ REG$(3): FOR N=17 TO 24: READ R1$(N): NEXT N
230 READ REG$(4): FOR N=25 TO 32: READ R1$(N): NEXT N
240 FOR N=1 TO 16: READ W1$(N): NEXT N 'WINNERS OF ROUND 1
250 FOR N=1 TO 16: READ R2$(N): NEXT N 'SECOND ROUND TEAMS
260 FOR N=1 TO 16: READ QF$(N): NEXT N 'REGIONAL QUARTERFINALS
270 FOR N=1 TO 8: READ SF$(N): NEXT N 'REGIONAL SEMIFINALS
280 FOR N=1 TO 4: READ FF$(N): NEXT N 'FINAL FOUR
290 FOR N=1 TO 2: READ NF$(N): NEXT N 'NATIONAL FINALS

```

```

300 READ NC$                                'NATIONAL CHAMPION
310 '***** PRINTING OF CHART HEADINGS *****
320 LPRINT CHR$(27);CHR$(31);                'BOLDFACE PRINTING
330 LPRINT P0$;"    FIRST";P1$;"    SECOND";P2$;"    REGIONAL";P3$;"    REG
    IONAL";P5$;"    NATIONAL";P6$;"    NATIONAL";CHR$(13);
340 LPRINT "    ROUND";P1$;"    ROUND";P2$;"    QUARTERFINALS";P3$;"    SEMIFI
    NALS";P4$;"    FINAL FOUR";P5$;"    FINALS";P6$;"    CHAMPION";CHR$(13);
350 GOTO 2000
1000 '***** DATA: TEAM LISTINGS, SCORES *****
1010 DATA EAST, SW LOUISIANA,RUTGERS,VIRGINIA COMMONW.,LASALLE,SYRACUSE,MOREHEAD
    STATE,WEST VIRGINIA,JAMES MADISON
1020 DATA WEST, WASHINGTON STATE,WEBER STATE,OKLAHOMA STATE,PRINCETON,N. CAROLIN
    A STATE,PEPPERDINE,ILLINOIS,UTAH
1030 DATA MIDEAST, TENNESSEE,MARQUETTE,PURDUE,ROBERT MORRIS,ILLINOIS STATE,OHIO
    U.,OKLAHOMA,ALABAMA-BIRMINGHAM
1040 DATA MIDWEST, MARYLAND,TENNESSEE-CHAT.,GEORGETOWN,ALCORN STATE,ALABAMA,LAMA
    R,IOWA,UTAH STATE
1050 DATA RUTGERS (60-53),VIRG.-COMM. (76-67),SYRACUSE (74-59),J. MADISON (57-50
    ),WASH. ST. (62-52),PRINCETON (56-53),N.C. STATE (69-67 20T),UTAH (52-49)
1060 DATA TENNESSEE (57-56),PURDUE (55-53),OHIO U. (51-49),OKLAHOMA (71-63),MARY
    LAND (52-51),GEORGETOWN (68-63),LAMAR (73-50),IOWA (64-59)
1070 DATA ST. JOHN'S,GEORGIA,OHIO STATE,NORTH CAROLINA,VIRGINIA,BOSTON COLLEGE,N
    EVADA-LAS VEGAS,UCLA,LOUISVILLE,ARKANSAS,KENTUCKY,INDIANA,HOUSTON,MEMPHIS S
    TATE,VILLANOVA,MISSOURI
1080 DATA ST. JOHN'S (66-55),GEORGIA (56-54),OHIO STATE (79-74),N. CAROLINA (68-
    49),VIRGINIA (54-49),BOSTON COLL.(51-42),N.C. STATE (71-70),UTAH (67-61)
1090 DATA LOUISVILLE (70-57),ARKANSAS (78-68),KENTUCKY (57-40),INDIANA (63-49),H
    OUSTON (60-50),MEMPHIS ST. (66-57),VILLANOVA (60-58),IOWA (77-63)
1100 DATA GEORGIA (70-67),N. CAROLINA (64-51),VIRGINIA (95-92),N.C. STATE (75-56
    ),LOUISVILLE (65-63),KENTUCKY (64-59),HOUSTON (70-63),VILLANOVA (55-54)
1110 DATA GEORGIA (82-77),N.C. STATE (63-62),LOUISVILLE (80-68),HOUSTON (89-71)
1120 DATA N.C. STATE (67-60),HOUSTON (94-81)
1130 DATA N.C. STATE (54-52)
2000 '***** REGIONAL PRINTING ROUTINE *****
2010 FOR Z=1 TO 4
2020 ON Z GOSUB 3000 , 4000 , 5000 , 6000
2030 FOR X=1 TO 2: LPRINT VU$;LU$;LF$;: NEXT X
2040 LPRINT P0$;PF$;R1$(1);NM$;VU$;LU$;LF$;
2050 LPRINT P0$;STRING$(17,241);CHR$(242);VU$;LU$;LF$;
2060 LPRINT P1$;CHR$(245);SP$;PF$;W1$(1);NM$;VU$;LU$;LF$;
2070 LPRINT P1$;CHR$(244);STRING$(17,241);CHR$(242);VU$;LU$;LF$;
2080 LPRINT P0$;PF$;R1$(2);NM$;P1$;CHR$(245);P2$;CHR$(245);SP$;PF$;QF$(1);NM$;VU
    $;LU$;LF$;
2090 LPRINT P0$;STRING$(17,241);CHR$(247);P2$;CHR$(244);STRING$(17,241);CHR$(242
    );VU$;LU$;LF$;
2100 LPRINT P1$;SP$;SP$;PF$;R2$(1);NM$;P2$;CHR$(245);P3$;CHR$(245);VU$;LU$;LF$;
2110 LPRINT P1$;STRING$(18,241);CHR$(247);P3$;CHR$(245);VU$;LU$;LF$;
2120 LPRINT P0$;PF$;R1$(3);NM$;P3$;CHR$(245);SP$;PF$;SF$(1);NM$;VU$;LU$;LF$;
2130 LPRINT P0$;STRING$(17,241);CHR$(242);P3$;CHR$(244);STRING$(17,241);CHR$(242
    );VU$;LU$;LF$;
2140 LPRINT P1$;CHR$(245);SP$;PF$;W1$(2);NM$;P3$;CHR$(245);P4$;CHR$(245);VU$;LU$
    ;LF$;
2150 LPRINT P1$;CHR$(244);STRING$(17,241);CHR$(242);P3$;CHR$(245);P4$;CHR$(245);
    VU$;LU$;LF$;

```

```

2160 LPRINT P0$;PF$;R1$(4);NM$;P1$;CHR$(245);P2$;CHR$(245);SP$;PF$;QF$(2);NM$;P3
    $;CHR$(245);P4$;CHR$(245);VU$;LU$;LF$;
2170 LPRINT P0$;STRING$(17,241);CHR$(247);P2$;CHR$(244);STRING$(17,241);CHR$(247
    );P4$;CHR$(245);VU$;LU$;LF$;
2180 LPRINT P1$;SP$;SP$;PF$;R2$(2);NM$;P2$;CHR$(245);P4$;CHR$(245);VU$;LU$;LF$;
2190 LPRINT P1$;STRING$(18,241);CHR$(247);P4$;CHR$(245);VU$;LU$;LF$;
2200 LPRINT P0$;PF$;R1$(5);NM$;P2$;STRING$(1,224);CHR$(15);HL$;PF$;REG$(2);" REG
    IONAL";HE$;CHR$(14);NM$;P4$;CHR$(245);SP$;PF$;FF$(1);NM$;VU$;LU$;LF$;
2210 LPRINT P0$;STRING$(17,241);CHR$(242);P4$;CHR$(244);STRING$(17,241);CHR$(E);
    LL$;LF$;
2220 LPRINT P1$;CHR$(245);SP$;PF$;W1$(3);NM$;P4$;CHR$(245);VL$;LL$;LF$;
2230 LPRINT P1$;CHR$(244);STRING$(17,241);CHR$(242);P4$;CHR$(245);VL$;LL$;LF$;
2240 LPRINT P0$;PF$;R1$(6);NM$;P1$;CHR$(245);P2$;CHR$(245);SP$;PF$;QF$(3);NM$;P4
    $;CHR$(245);VL$;LL$;LF$;
2250 LPRINT P0$;STRING$(17,241);CHR$(247);P2$;CHR$(244);STRING$(17,241);CHR$(242
    );P4$;CHR$(245);VL$;LL$;LF$;
2260 LPRINT P1$;SP$;SP$;PF$;R2$(3);NM$;P2$;CHR$(245);P3$;CHR$(245);P4$;CHR$(245)
    ;VL$;LL$;LF$;
2270 LPRINT P1$;STRING$(18,241);CHR$(247);P3$;CHR$(245);P4$;CHR$(245);VL$;LL$;LF
    $;
2280 LPRINT P0$;PF$;R1$(7);NM$;P3$;CHR$(245);SP$;PF$;SF$(2);NM$;P4$;CHR$(245);VL
    $;LL$;LF$;
2290 LPRINT P0$;STRING$(17,241);CHR$(242);P3$;CHR$(244);STRING$(17,241);CHR$(247
    );VL$;LL$;LF$;
2300 LPRINT P1$;CHR$(245);SP$;PF$;W1$(4);NM$;P3$;CHR$(245);VL$;LL$;LF$;
2310 LPRINT P1$;CHR$(244);STRING$(17,241);CHR$(242);P3$;CHR$(245);VL$;LL$;LF$;
2320 LPRINT P0$;PF$;R1$(8);NM$;P1$;CHR$(245);P2$;CHR$(245);SP$;PF$;QF$(4);NM$;P3
    $;CHR$(245);VL$;LL$;LF$;
2330 LPRINT P0$;STRING$(17,241);CHR$(247);P2$;CHR$(244);STRING$(17,241);CHR$(247
    );VL$;LL$;LF$;
2340 LPRINT P1$;SP$;SP$;PF$;R2$(4);NM$;P2$;CHR$(245);VL$;LL$;LF$;
2350 LPRINT P1$;STRING$(18,241);CHR$(247);VL$;LL$;LF$;
2360 FOR X=1 TO 2: LPRINT P5$;VL$;LL$;LF$;: NEXT X
2370 LPRINT VL$;SP$;PF$;NF$(1);NM$;LL$;LF$;
2380 LPRINT P5$;CHR$(J);STRING$(17,K);CHR$(L);C$;LF$;
2390 NEXT Z
2400 PRINT "JOB COMPLETED": END
3000 '***** EAST REGION CONTENTS *****
3010 FOR Y=1 TO 8: R1$(Y)=R1$(Y): NEXT Y
3020 FOR X=1 TO 4: W1$(X)=W1$(X): R2$(X)=R2$(X): QF$(X)=QF$(X): NEXT X
3030 FOR W=1 TO 2: SF$(W)=SF$(W): NEXT W
3040 FF$(1)=FF$(1): NF$(1)=NF$(1)
3050 VU$="": LU$="": VL$=P5$+CHR$(245): LL$=""
3060 I=7: E=242: J=244: K=241: L=242: C$=""
3070 RETURN
4000 '***** WEST REGION CONTENTS *****
4010 FOR Y=1 TO 8: R1$(Y)=R1$(Y+8): NEXT Y
4020 FOR X=1 TO 4: W1$(X)=W1$(X+4): R2$(X)=R2$(X+4): QF$(X)=QF$(X+4): NEXT X
4030 FOR W=1 TO 2: SF$(W)=SF$(W+2): NEXT W
4040 FF$(1)=FF$(2): NF$(1)=""
4050 VU$=P5$+CHR$(245): LU$=P6$+CHR$(245): VL$="": LL$=P6$+CHR$(245)
4060 I=7: E=247: J=224: K=224: L=224: C$=P6$+CHR$(244)+CHR$(20)+CHR$(27)+CHR$(30
    )+CHR$(27)+CHR$(30)+SP$+PF$+NC$+NM$+LF$+LF$+P6$+STRING$(17,241)
4070 RETURN

```

```

5000 '***** MIDEAST REGION CONTENTS *****
5010 FOR Y=1 TO 8: R1$(Y)=R1$(Y+16): NEXT Y
5020 FOR X=1 TO 4: W1$(X)=W1$(X+8): R2$(X)=R2$(X+8): QF$(X)=QF$(X+8): NEXT X
5030 FOR W=1 TO 2: SF$(W)=SF$(W+4): NEXT W
5040 FF$(1)=FF$(3): NF$(1)=NF$(2)
5050 VU$="": LU$=P6$+CHR$(245): VL$=P5$+CHR$(245): LL$=P6$+CHR$(245)
5060 I=5: E=242: J=244: K=241: L=247: C$=""
5070 RETURN
6000 '***** MIDWEST REGION CONTENTS *****
6010 FOR Y=1 TO 8: R1$(Y)=R1$(Y+24): NEXT Y
6020 FOR X=1 TO 4: W1$(X)=W1$(X+12): R2$(X)=R2$(X+12): QF$(X)=QF$(X+12): NEXT X
6030 FOR W=1 TO 2: SF$(W)=SF$(W+6): NEXT W
6040 FF$(1)=FF$(4): NF$(1)=""
6050 VU$=P5$+CHR$(245): LU$="": VL$="": LL$=""
6060 I=5: E=247: J=224: K=224: L=224: C$=""
6070 RETURN

```

At no point does the TRNYGRAM program enter the bit-image graphic mode, but our next illustration mixes the built-in block graphic characters with many elements drawn in bit graphics. This seems to be necessary in architectural drawing, in order to provide enough detail and accuracy.

Figure 11-4 shows the printout of an architectural plan of a house in Madison, Wisconsin. The program for it, called MADHOUSE (which is entirely appropriate as deadlines near), is listed only partially (program 11-2).

This program uses three of the ready-made rectangular block characters (codes 239, 231, and 232) on the DMP-400 for the walls and door frames of the plan and triangular characters (252 and 254) for part of the fireplace. It also makes a great deal of use of the six-dot-wide blank, CHR\$(224), for the many empty spaces in the drawing.

These characters and $\frac{9}{32}$ -inch (or $\frac{1}{12}$ -inch) line feeds in text mode are continually intermixed with nearly two dozen bit-image strings for such features as patio doors (line 150), different sizes of windows (lines 160, 170, 420, and 430), and single-dot lines (used for the stairway, deck, kitchen counters, and a built-in desk). Curved lines in the bathroom fixtures and dotted lines for closets are also drawn in the bit-image mode.

The orientation of the drawing has the longest dimension of the house parallel to the printhead movement, so that built-in alphabets can be used for the labeling. Carriage returns are handled by the direct-positioning sequence (27+16+0+0) defining CR\$ in line 30, to guarantee that

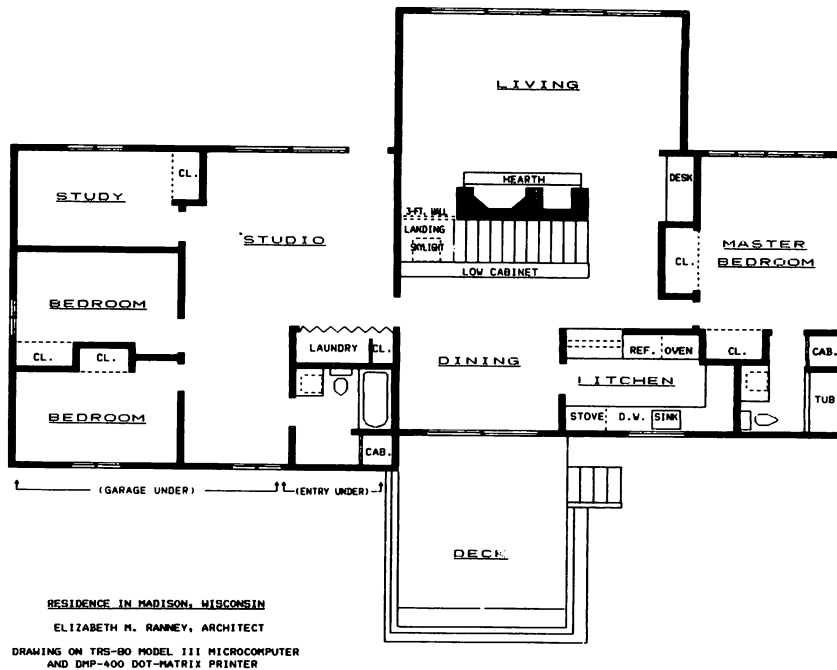


FIGURE 11-4. Using a mixture of block-graphic characters and bit-image drawing: an architectural plan (MADHOUSE program).

each print line starts at the left margin and that carriage returns are independent of line feeds.

In the programming it is essential to have an accurate drawing or tracing of the building. If a standard plan having a scale of $\frac{1}{4}$ -inch per foot is traced with eight-squares-per-inch, see-through graph paper, each square will represent six inches by six inches of the real building, and each dot in the six-by-six matrix of a block graphic character will represent exactly one inch. This makes it easier to figure out the coding for in-between sizes and distances that are not even multiples of six. All this assumes another essential procedure—using elite (12-CPI) rather than pica (10-CPI) dot density for the whole drawing, so that horizontal distances will equal vertical distances.

PROGRAM 11-2. MADHOUSE. Architectural drawing combining the block-graphic and bit-image approaches.

```
10 'PROGRAM 11-2: "MADHOUSE" -- ARCHITECTURAL DRAWING
13 'Residential architectural plan: Elizabeth M. Ranney, Architect, Madison, WI
15 'FOR TRS-80 COMPUTERS AND WIDEBODY PRINTERS
20 CLEAR 1000
30 CR$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0) 'CARRIAGE RETURN
```

```

40 PN$=CHR$(27)+CHR$(16)                                ' POSITIONING STRING
50 P0$=CHR$(27)+CHR$(16)+CHR$(0)                        '
60 P1$=CHR$(27)+CHR$(16)+CHR$(1)                        '
70 P2$=CHR$(27)+CHR$(16)+CHR$(2)                        '
80 CF$=CHR$(27)+CHR$(18)                                ' CORRESPONDENCE FONT
90 NM$=CHR$(27)+CHR$(23)                                ' NORMAL 12-CPI DENSITY
95 CP$=CHR$(27)+CHR$(20)                                ' CONDENSED 16.7-CPI DENSITY
100 LF$=CHR$(30)+CHR$(20)+CHR$(27)+CHR$(28)+CHR$(19)    ' 1/12" LF
110 RLF$=CHR$(30)+CHR$(20)+CHR$(27)+CHR$(30)+CHR$(19)   ' REVERSE LF
120 LPRINT CHR$(30);NM$;CHR$(27);CHR$(31);              ' BOLD PRINTING
130 LB$=CHR$(30)+CHR$(27)+CHR$(32)+CHR$(15)+CHR$(27)+CHR$(14)+CF$ ' ELONGATED
    PRINTING W/ UNDERLINE
140 UL$=CHR$(14)+CHR$(27)+CHR$(15)+NM$+CHR$(27)+CHR$(31) ' END ELONGATED
    & UNDERLINE
145 '***** BUILDING BLOCKS *****
150 PD$=CHR$(18)+STRING$(47,165)+STRING$(2,191)+STRING$(47,169)+CHR$(30)
    ' PATIO DOOR
160 W$=CHR$(18)+STRING$(23,169)+STRING$(2,191)+STRING$(23,169)+CHR$(30)
    ' DOUBLE WINDOW
170 PW$=CHR$(18)+STRING$(46,169)+STRING$(4,191)+STRING$(46,169)+CHR$(30)
    ' PATIO WINDOW
180 SL$=CHR$(18)+CHR$(191)+STRING$(5,128)+CHR$(30)      ' SOLID LINE (+5 BLANK DOT-P
    OSITIONS)
190 DL$=CHR$(18)+CHR$(156)+STRING$(5,128)+CHR$(30)      ' DASHED LINE (+ 5 BLANK DOT
    -POSITIONS)
200 LD$=CHR$(18)+STRING$(4,128)+CHR$(156)+CHR$(128)+CHR$(30) ' DASHED
    LINE (W/ 4 BLANKS BEFORE, 1 AFTER)
210 LS$=CHR$(18)+STRING$(5,128)+CHR$(255)+CHR$(30)      ' SOLID LINE (PRECEDED BY
    5 BLANK DOT POSITIONS)
220 EW$=CHR$(30)+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(36)+CHR$(239) ' EAST WALL POS.
.
.
.
380 TS$=CHR$(18)+CHR$(191)+STRING$(3,128)+CHR$(191)+CHR$(128)+CHR$(30)
    ' TUB SIDE (EAST)
390 TW$=CHR$(18)+CHR$(128)+CHR$(191)+STRING$(3,128)+CHR$(191)+CHR$(30)
    ' TUB SIDE (WEST,WALL)
400 TT$=CHR$(18)+STRING$(6,136)+CHR$(30)                ' TUB TOP
410 LL$=EW$+CHR$(27)+CHR$(16)+CHR$(0)+CHR$(180)+CHR$(239) ' NORTHEAST BE
    DROOM WALLS
420 KW$=CHR$(18)+STRING$(48,165)+CHR$(30)                ' KITCHEN WINDOW
430 NW$=CHR$(18)+STRING$(20,165)+STRING$(2,191)+STRING$(20,165)+CHR$(30)
    ' OTHER NORTH WINDOWS
440 DD$=CHR$(27)+CHR$(16)+CHR$(2)+CHR$(5)+CHR$(18)+CHR$(191)+CHR$(30)
    ' DECK, WEST EDGE, NEXT TO DINING ROOM DOOR
450 DW$=DD$+LS$+CHR$(224)+LS$                            ' MAIN WEST DECK EDGE
460 DE$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(104)+SL$+BS$     ' MAIN EAST DECK EDGE
470 RO$=CHR$(27)+CHR$(16)+CHR$(2)+CHR$(85)+CHR$(18)+CHR$(156)+CHR$(30) ' DASHE
    D LINE BETWEEN REFRIG. & OVEN
480 SV$=CHR$(27)+CHR$(16)+CHR$(2)+CHR$(38)+CHR$(18)+CHR$(156)+CHR$(30) ' DASHE
    D LINE BETWEEN STOVE & DISHWASHER
490 '***** START OF DRAWING *****
500 LPRINT P1$;CHR$(110);STRING$(5,239);PD$;CHR$(239);PD$;STRING$(4,239);LF$;
    ' SOUTH LIVING ROOM WALL

```

```

510 FOR N=2 TO 24: LPRINT P1$;CHR$(110);CHR$(239);P2$;CHR$(100);CHR$(239);: LPRINT CHR$(224);LF$;: NEXT N      'EAST & WEST LIVING ROOM WALLS
520 FOR N=1 TO 12: LPRINT RLF$;: NEXT N: LPRINT P1$;CHR$(198);LB$;"LIVING";UL$;
530 FOR N=1 TO 15: LPRINT LF$;: NEXT N: LPRINT P1$;CHR$(190);CF$;"HEARTH";NM$;
540 FOR N=1 TO 5: LPRINT RLF$;: NEXT N
550 FOR N=1 TO 3: LPRINT CHR$(27);CHR$(50);: NEXT N      '3/72" SPACE
560 LPRINT CR$;STRING$(6,224);STRING$(8,239);W$;STRING$(16,239);PW$;CHR$(233);CHR$(8);CHR$(6);CHR$(18);STRING$(33,128);CHR$(30);CHR$(239);CHR$(239);STRING$(37,224);STRING$(7,239);W$;CHR$(233);CHR$(8);CHR$(6);W$;STRING$(3,239);CHR$(233);LF$;
570 FOR N=1 TO 3: LPRINT CR$;EW$;STRING$(22,224);DL$;STRING$(3,224);CHR$(239);STRING$(27,224);CHR$(239);DK$;STRING$(3,224);CHR$(239);WW$;LF$;: NEXT N
580 LPRINT P0$;CHR$(180);CF$;"CL.";NM$;P2$;CHR$(92);CP$;"DESK";NM$;
590 LPRINT EW$;STRING$(22,224);DL$;STRING$(3,224);CHR$(239);STRING$(27,224);CHR$(239);STRING$(9,224);CHR$(18);CHR$(255);STRING$(100,129);CHR$(255);
600 LPRINT DK$;STRING$(3,224);CHR$(239);WW$;LF$;
610 LPRINT EW$;STRING$(22,224);DL$;STRING$(3,224);CHR$(239);STRING$(27,224);CHR$(239);STRING$(9,224);CHR$(18);CHR$(255);STRING$(100,160);CHR$(255);CHR$(30);
620 LPRINT DK$;STRING$(3,224);CHR$(239);WW$;LF$;
630 LPRINT EW$;STRING$(22,224);DL$;STRING$(3,224);CHR$(239);STRING$(27,224);CHR$(239);STRING$(8,224);STRING$(2,232);BS$;SL$;CHR$(227);STRING$(7,224);STRING$(2,232);CHR$(227);STRING$(3,224);CHR$(228);
640 LPRINT CHR$(232);CHR$(232);BS$;CHR$(8);CHR$(3);SL$;DK$;STRING$(3,224);CHR$(239);WW$;LF$;
650 LPRINT EW$;STRING$(22,224);DL$;STRING$(3,224);CHR$(239);STRING$(27,224);CHR$(239);STRING$(8,224);STRING$(2,239);CHR$(233);STRING$(7,224);STRING$(2,239);CHR$(233);STRING$(3,224);CHR$(234);STRING$(2,239);
660 LPRINT DK$;STRING$(3,224);CHR$(239);WW$;LF$;
670 LPRINT P0$;CHR$(75);LB$;"STUDY";UL$;
.
.
.
1440 FOR N=1 TO 2: LPRINT DE$;LS$;LS$;DW$;LF$;: NEXT N
1450 LPRINT DE$;CHR$(224);CHR$(8);CHR$(2);CHR$(18);CHR$(255);STRING$(6,192);STRING$(146,193);STRING$(5,192);CHR$(255);STRING$(11,128);CHR$(191);LF$;
1460 LPRINT DE$;CHR$(18);P2$;CHR$(23);CHR$(191);LF$;
1470 LPRINT P1$;CHR$(104);CHR$(30);CHR$(8);CHR$(4);CHR$(246);STRING$(28,241);CHR$(27);CHR$(2);CHR$(247);LF$;
1480 FOR N=1 TO 16: LPRINT RLF$;: NEXT N: LPRINT P1$;CHR$(164);LB$;"DECK";UL$;
1490 FOR N=1 TO 11: LPRINT RLF$;: NEXT N
1510 LPRINT CR$;STRING$(7,224);CHR$(246);STRING$(9,241);CHR$(224);CF$;"(GARAGE UNDER)";NM$;CHR$(224);STRING$(8,241);CHR$(8);CHR$(1);CHR$(247);
1520 LPRINT CHR$(246);"(ENTRY UNDER)";CHR$(27);CHR$(4);CHR$(247);LF$;
1530 FOR N=1 TO 14: LPRINT LF$;: NEXT N
1540 LF$=CHR$(13)
1550 LPRINT CHR$(27);CHR$(31);P0$;CHR$(60);CHR$(15);"RESIDENCE IN MADISON, WISCONSIN";CHR$(14);LF$;LF$;
1560 LPRINT STRING$(9,224);CHR$(27);CHR$(31);"ELIZABETH M. RANNEY, ARCHITECT";LF$;LF$;
1570 LPRINT STRING$(5,224);"DRAWING ON TRS-80 MODEL III MICROCOMPUTER";LF$;
1580 LPRINT STRING$(11,224);"AND DMP-400 DOT-MATRIX PRINTER";LF$;

```

Once the strings have been defined, the MADHOUSE program just goes right ahead with the drawing, starting with the living room's outside wall with patio doors. Occasionally reverse line feeds and backspaces make the drawing more convenient, especially the labeling. (Those wanting to do architectural drawings on the MX- or FX-100 printers, which have no reverse line feed, can get around the problem by interrupting the drawing of lines wherever a label needs to be printed.) From the start of the study and studio sections, the program usually takes one line of instructions for each line that it prints. About 105 sweeps of the printhead, taking around eight minutes, are needed for printing the entire drawing.

Unfortunately, there are no READ-and-DATA or datafile inputs in this type of program—it just draws what the program statements tell it to, line by line. Considering the huge differences among houses, it's hard to imagine what form a more general program would take, in residential architecture at least. Still, there may be good reasons for having these computer-drawn plans like the one drawn by MADHOUSE. Besides providing as many copies as you may need of a finished plan, there is plenty of room for making changes in a plan that is only in the rough-draft stage.

There may be more use for microcomputer-drawn plans outside of residential architecture. If a building is, say, a multistory commercial structure with many floors having similar plans, then one basic plan worked out on the computer could, with minor changes as needed, serve for all floors of the building. Furthermore, as dot-matrix printers improve in their printing quality in the near future, it's possible that even a skilled draftsman can arrive at a neater and more accurate drawing using a microcomputer than doing it by hand. And after some practice involving the same graphic codes over and over, draftsmen and draftswomen might find that it's faster than by hand. (But plan on ten to twenty hours for a single drawing the first time you try it!)

Let's keep our perspective, though. What we're doing here with a microcomputer is a far cry from the fifty-thousand-dollar-and-up CAD/CAM systems used by large architectural and engineering firms. It's also a country mile from the five-hundred-dollar software packages for computer-aided

design that have recently come into the microcomputer marketplace.

Bit-Image Diagrams and Maps

The block and line-graphic characters in the printer's ROM are basically low-resolution tools. When we move on to drawings that are done entirely in the bit-graphic mode, we'll find a host of better possibilities.

Maps. Figure 11-5 provides our first example—a map of the Lower 48 United States, drawn by a program called USAMAP.

Actually, the program that drew this map is none other than our old friend, LINEDRAW (see chapters 5 and 6). In fact it is identical to the LINEDRAW type of program used in the drawing of Dennis the Menace (figure 1-2), apart from DATA content.

But it was certainly a bigger project. After placing a grid

FIGURE 11-5. A map created by a LINEDRAW-type program called USAMAP (DMP-400).



(again, thin graph paper) over an eleven-by-fourteen-inch map of the United States from a road atlas, a coding sheet running forty-four by eighteen inches was prepared by gluing together several sheets of ten-squares-per-centimeter opaque graph paper. Using the ratio of ten horizontal squares for every seven vertical squares for condensed printing on the DMP-400, the state boundaries and coastlines were drawn on the coding sheet by hand and eye from the grid over the atlas map. Then the printing passes and DATA lines were defined by inked horizontal lines drawn on every seventh line of the coding sheet, creating ninety rectangular minigrids (each seven squares high and ten squares wide) in each of fifty-two rows. The dot patterns for the ten columns in each of these 4,680 minigrids then had to be figured out and entered into the DATA lines of the computer program. So as to increase the chances of these extended efforts doing somebody some good, all 46,800 codes are provided in appendix A (program A-2, USAMAP).

Some words of caution to anyone thinking of tackling a coding task of this size. Don't use ten-squares-per-centimeter graph paper—that's about twenty-five squares per inch. Even with the most perfect vision and a magnifying glass, it's just too difficult to distinguish the dot columns. Be advised not to try anything smaller than ten-squares-per-inch graph paper, in fact. Although your coding sheet may have to be eight feet long, you can save hours—perhaps half a week—by making fewer mistakes in the coding. With ten squares to the inch, you have room enough in each column of a minigrid to write the dot code for that column with a sharp pencil, and this can be a godsend in difficult code sequences such as those for the Texas coastline.

Other tips: Use thin vertical lines on both sides of the drawing, or at the very least on the right-hand side, so that you can keep checking whether the number of codes in any row is correct without having to count them. Better yet, put a byte counter into the program like those in the LINEDRAW, FOTOGRAF, or PICTOGRM programs. And when you find that the state boundary lines in one row are not lining up the way they should with lines in the row just above or below, explode the map in a black-white reversal, that is, add a few $\frac{1}{216}$ -inch line feeds to the standard $\frac{7}{72}$ -inch feed to separate the rows, and then print the map white-on-black in standard

or elite density (as space on the paper permits). This will usually reveal rather quickly where (within a row) you've made a coding error.

A further suggestion: Divide each row into small, equal sections, say, of thirty dot columns apiece, and check your count of codes for each section that has been keyed into the program's DATA lines, especially if you're using an IBM-Epson printer, to increase the chances of coming out with exactly the right total at the end of each whole print line. Also, divide the whole project into several segments, numbering your DATA statements accordingly, as we did with WHITECAT (see figure 5-21 and its explanation in the text). You'll find it easier to do the coding in binary sums ranging from 0 to 127 no matter what kind of printer you have (unless it has fewer than seven addressable pins). And if it's a TRS-80 requiring dot codes in the 128 to 255 range, leave it to your LINEDRAW program to add 128 to binary-sum codes (or for black-white reversal, to subtract the codes from 255) just before the LPRINT command is executed. Also, if you're tempted to use rows that are eight dots high on the IBM or Epson, remember how much harder it is to add the binaries in your head when you have to cope with the whole range of 0 to 255; the number of rows you have to code is somewhat smaller, but the price you pay for that advantage could be that it takes three times longer to do the adding and that there are five times as many errors in the coding.

Complex bit-image diagrams. A more challenging task is to duplicate the kind of hand-drawn diagram that the author introduced in *Baseball Graphics* (First Impressions, 1979)—a diagram of a baseball game, called a “gamegram.” Figure 11-6 shows a gamegram drawn by a Radio Shack LPVIII printer that is virtually identical to the original hand-drawn diagrams in that book.

The lesson learned with this project was that even some diagrams you don't think can be done with a microcomputer and printer usually can be. To simulate the by-hand gamegram, it was necessary to write a bunch of subroutines that print only one symbol apiece—a black square for an out, an open square for a single, a double-sized rectangle for a two-base hit, a quadruple-sized rectangle for a home run (which

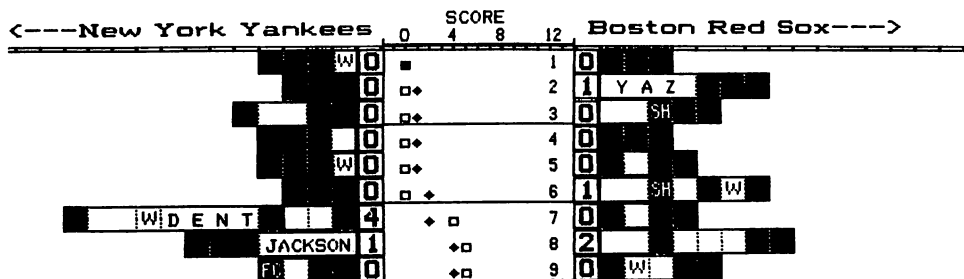


FIGURE 11-6. Diagram of a baseball game (the 1978 American League East playoff game) drawn by the GAMEGRAM program. Abbreviations: black square = an out; white square = single; white rectangle = two-base hit; larger rectangle containing name = home run; W = walk; SH = sacrifice hit; FC = fielder's choice. Read action sequence left to right for Boston and right to left for New York.

is big enough to enclose the name of the slugger who hit it), and so forth. The sequence of action within an inning (which should be read left to right for the team on the right and right to left for the team on the left) can be coded with abbreviations such as SOODWO for the sequence single-out-out-double-walk-out, using reversed order for the visiting team on the left. Then, for each sequence, an IF-(action code)-THEN statement can direct the program to the appropriate sequence of subroutines to print each inning's action sequence in a series of symbols. The length of each series varies roughly with the total-bases statistic, so the overall diagram turns out to be much like a two-way horizontal bar graph.

The inning-by-inning score of the game is printed in the center of the diagram, like the original hand-drawn version, except for omitting the connecting lines between the data points.

The program, called GAMEGRAM, which draws figure 11-6 will be presented fully elsewhere. The game itself will be remembered by Boston Red Sox fans as the worst outcome of a season in that team's history—the 1978 playoff game against the New York Yankees in which the Red Sox were beaten by their famous Green Monster, Fenway Park's left-field wall (more specifically, by Bucky Dent's fly ball that went over that wall). GAMEGRAM is designed to be a program that will draw any baseball game, even a twenty-five inning one, depending on what is put into the DATA lines.

Program 11-3 shows the set of DATA statements for this play-off game.

The number of subroutine sequences needed to cover the dozens of different action sequences that can occur in an inning can get to be pretty substantial in this kind of program. In any nine-inning game, there won't be more than eighteen different sequences needed (nine for each team), and it usually runs closer to a dozen because "out 1-2-3" innings are common. But as an example of what's needed over many games, consider the seven-game World Series in 1982 (see figure 12-6, last chapter): Forty-nine different subroutine sequences were needed to cover the inning-by-inning offensive output by the St. Louis Cardinals, and fifty-six were needed for the Milwaukee Brewers. In the further development of this program, some sort of sequence-assembly routine might replace all of these subroutines, even for sequences such as the Cardinals' OESODWSSWDWO in Game 4's sixth inning. Programmers who love to say "there's always a way, or better way" could have fun with this problem.

PROGRAM 11-3. GAMEGRAM/DAT. Bit-image diagram of a baseball game (DATA statements only).

```
10 'PROGRAM 11-3: "GAMEGRAM/DAT" -- DATA FOR BASEBALL GAME DIAGRAM ('GAMEGRAM')
1000 REM YANKEES 5, RED SOX 4 (OCT. 2,1978); DATA ARE INNING,VISTM RUNS,VISTM PL
    AYS,HOMETM RUNS,HOMETM PLAYS
1010 DATA 1,0,W000,0,000
1020 DATA 2,0,000,1,H000
1030 DATA 3,0,00D0,0,DB00
1040 DATA 4,0,S000,0,000
1050 DATA 5,0,W000,0,0S00
1060 DATA 6,0,000,1,DBSOW0
1070 DATA 7,4,0SSOHWD0,0,0S00
1080 DATA 8,1,H000,2,D0SSS00
1090 DATA 9,0,00SF,0,0WS00
```

Tables

We usually think of tables as being dull columns of numbers, devoid of any visual interest. But it's possible that by injecting some graphic features in the construction of tables we can perk them up visually and make them more informative, too. This is the topic of exploration in the rest of this chapter.

Both typewriters and dot-matrix printers can be used for producing unlined tables that consist merely of columns of

numbers under column headings and a general title. Any typist will tell you that this is more bother than typing regular text with a typewriter—some fussing around is needed to space the columns neatly, line up the decimal points vertically, and so forth. But that probably can be done more quickly than programming a table from scratch with the computer and printer, even with such conveniences of the latter as TAB(), vertical tab-sets, print punctuation (using commas instead of semicolons) for formatting the table, and that wonderful LPRINT USING command for lining up decimal points.

If you want a table that is lined horizontally and vertically, however, the way they often come out in the published product, the computer and printer have a clear advantage over the typewriter or even drawing inked lines by hand. And if you have several dozen or several hundred lined tables of the same type to do, there's no contest.

Lined tables. Programming thin horizontal lines in a table is no problem. Without even going into graphic mode, you can repeat the horizontal line-graphic character (code 241) on Radio Shack printers and the corresponding characters (code 196 or 205) on IBM printers. For dashed lines, the simplest way is to repeat the hyphen (ASCII code 45) itself. Among many options you have in graphic mode are printing thicker lines (repeating dot code 28 or 56), double lines (repeating dot code 24), and dotted lines (repeating sequences of dot printing and blank codes).

For vertical lines, the TRS-80's line-graphic code 245 combined with 1/12-inch line feeds will handle things in text mode. And in graphic mode the top six dots on the Tandy or Apple printers (dot code 191) or the middle six dots on IBM-Epson printers (dot code 63) will give you continuous vertical lines if the same line-feed size is used. Thicker, double, and dotted vertical lines are just as easy to program (see figure 5-3).

The accounting table in figure 11-7, from a program called ACCTABLE (program 11-4), uses the DMP-400's built-in characters 241 and - for horizontal lines and the bit-graphic sequence CHR\$(201);CHR\$(128);CHR\$(201) for double-dotted verticals. The program is arranged to read various assortments of account names, titles, column headings, line

SCINTILLATING SCIENTIFIC INSTITUTE

PAST-PRESENT-FUTURE INCOME SUMMARY (1981-1984) AS OF 5/4/83

SOURCE OF INCOME	RECEIVED 1981-82	IN HAND 1982-83	PENDING 1982-83	PROBABLE 1983-84	PENDING 1983-84
UNIT 1					
Grant NIH-33982-03	13,807	8,000	-----	13,500	-----
Grant NSF-00235-01	-----	22,500	-----	25,500	-----
Grant MOD-9458X-03	1,200	1,400	-----	1,600	-----
UNIT 2					
Grant DOD-10345-01	-----	29,000	-----	33,000	-----
Grant VAA-87654-02	8,500	9,000	-----	9,500	-----
UNIT 3					
Grant ACS-97531-01	-----	3,400	-----	4,000	-----
Univ. support grant	-----	1,550	-----	-----	-----
UNIT 4					
NIH-98764-03	13,000	-----	-----	-----	-----
NIMH-23456	-----	-----	22,000	-----	24,000
NSF-00167	-----	-----	21,000	-----	23,500
DOD-98897	-----	-----	17,500	-----	20,000
OTHER INCOME					
Equipment Sales	5,500	4,300	1,200	5,000	-----
Animal Sales	9,856	3,305	4,000	-----	7,200
Miscell. Per Diems	2,300	3,800	1,000	5,000	3,000
TOTALS:	\$ 54,163	\$ 86,255	\$ 66,700	\$ 97,100	\$ 77,700

FIGURE 11-7. A line table (ACCTABLE program).

identifications, dollar amounts, and blanks (zeros printed as ---) for DATA statements.

It also adds up the figures in each column to provide printed totals. Most of the positioning of names or numbers is accomplished by the TAB() and LPRINT USING commands. So that zeros will be printed unslashed, the DMP-400's correspondence font is used for any printing of numbers.

PROGRAM 11-4. ACCTABLE. A lined table for accounting.

```

10 'PROGRAM 11-4: "ACCTABLE" -- ACCOUNTING TABLE
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 III OR 4 AND 8- OR 15-INCH TRS-80 PRINTERS
30 CLEAR 1000
40 INPUT "NAME OF ACCOUNT";ACCT$
50 Y=240-6*LEN(ACCT$)
100 FOR N=1 TO 2: LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(Y);CHR$(27);CHR$(14);CHR$(15);ACCT$;CHR$(14);CHR$(27);CHR$(15);: NEXT N 'PRINT TITLE (ACC'T NAME)
110 LPRINT STRING$(2,13);CHR$(27);CHR$(31);
120 LPRINT TAB(17);"PAST-PRESENT-FUTURE INCOME SUMMARY (1981-1984)";" ";CHR$(15);"AS OF 5/4/83";CHR$(14);CHR$(10)
130 LPRINT TAB(29);"RECEIVED";TAB(40);"IN HAND";TAB(50);"PENDING";TAB(59);"PROBABLE";TAB(70);"PENDING"
140 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(19)

```

```

150 LPRINT CHR$(15) 'SET UNDERLINE, LINE FEED
160 LPRINT "SOURCE OF INCOME";TAB(30);"1981-82";TAB(40);"1982-83";TAB(50);"1982-
    83";TAB(60);"1983-84";TAB(70);"1983-84 ";
170 LPRINT CHR$(14);CHR$(20);CHR$(27);CHR$(56);CHR$(19); 'END UNDERLINE
180 GOSUB 470 'START OF VERTICAL DIVIDERS AND CORRESPONDENCE-QUALITY FONT
190 READ U$
200 IF U$="H" THEN READ MH$: LPRINT MH$;: GOSUB 470 : GOTO 190
210 IF U$="EU" THEN GOSUB 560 : GOTO 190
220 PRINT STRING$(63,"*")
230 LPRINT " ";U$;: READ A1,A2,A3,A4,A5: IF A1=0 THEN LPRINT TAB(31);"-----";:
    GOTO 270
240 IF A1=0 THEN LPRINT TAB(31);"-----";: GOTO 270
250 PF$="##,###"
260 LPRINT TAB(31);USING PF$;A1;: T1=T1+A1: PRINT T1;
270 IF A2=0 THEN LPRINT TAB(41);"-----";: GOTO 290
280 LPRINT TAB(41);USING PF$;A2;: T2=T2+A2: PRINT T2;
290 IF A3=0 THEN LPRINT TAB(51);"-----";: GOTO 310
300 LPRINT TAB(51);USING PF$;A3;: T3=T3+A3: PRINT T3
310 IF A4=0 THEN LPRINT TAB(61);"-----";: GOTO 330
320 LPRINT TAB(61);USING PF$;A4;: T4=T4+A4: PRINT T4;
330 IF A5=0 THEN LPRINT TAB(71);"-----";: GOSUB 470 : GOTO 350
340 LPRINT TAB(71);USING PF$;A5;: T5=T5+A5: PRINT T5: PRINT: GOSUB 470
350 ON ERROR GOTO 370
360 GOTO 190
370 LPRINT CHR$(30);CHR$(27);CHR$(19);STRING$(78,241);CHR$(27);CHR$(18);CHR$(13)
    ;
380 TF$="$###,###"
390 LPRINT TAB(13);CHR$(15);"TOTALS:";CHR$(14);TAB(31);USING TF$;T1;
400 LPRINT TAB(41);USING TF$;T2;
410 LPRINT TAB(51);USING TF$;T3;
420 LPRINT TAB(61);USING TF$;T4;
430 LPRINT TAB(71);USING TF$;T5
440 LPRINT CHR$(30);CHR$(27);CHR$(19);CHR$(27);CHR$(32);
450 END
460 '***** VERTICAL DIVIDERS *****
470 A$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(168)+CHR$(201)+CHR$(128)+CHR$(201)
480 B$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(228)+CHR$(201)+CHR$(128)+CHR$(201)
490 C$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(32)+CHR$(201)
500 D$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(92)+CHR$(201)+CHR$(128)+CHR$(201)
510 E$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(152)+CHR$(201)
520 F$=CHR$(27)+CHR$(16)+CHR$(1)+CHR$(212)+CHR$(201)+CHR$(128)+CHR$(201)
530 LPRINT CHR$(30);CHR$(27);CHR$(19);CHR$(18);A$;B$;C$;D$;E$;F$;CHR$(30);CHR$(2
    7);CHR$(18);
540 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(27);CHR$(30);CHR$(27);CHR$(5
    6);CHR$(19);: LPRINT CHR$(27);CHR$(50);
550 LPRINT CHR$(13);: RETURN
560 '***** HORIZONTAL DIVIDER *****
570 LPRINT CHR$(30);STRING$(78,"-"); CHR$(13);
580 RETURN
590 DATA H,UNIT 1
600 DATA Grant NIH-33982-03,13807,8000,0,13500,0
610 DATA Grant NSF-00235-01,0,22500,0,25500,0
620 DATA Grant MOD-9458X-03,1200,1400,0,1600,0,EU

```



```

630 DATA H,UNIT 2
640 DATA Grant DOD-10345-01,0,29000,0,33000,0
650 DATA Grant VAA-87654-02,8500,9000,0,9500,0,EU
660 DATA H,UNIT 3
670 DATA Grant ACS-97531-01,0,3400,0,4000,0
680 DATA Univ. support grant,0,1550,0,0,0,EU
690 DATA H,UNIT 4
700 DATA NIH-98764-03,13000,0,0,0,0
710 DATA NIMH-23456,0,0,22000,0,24000
720 DATA NSF-00167,0,0,21000,0,23500
730 DATA DOD-98897,0,0,17500,0,20000,EU
740 DATA H, OTHER INCOME
750 DATA Equipment Sales,5500,4300,1200,5000,0
760 DATA Animal Sales,9856,3305,4000,0,7200
770 DATA Miscell. Per Diems,2300,3800,1000,5000,3000

```

Column-centered tables. The ACCTABLE program positions column headings with tab stops so that they are centered at the head of each column, but a more sophisticated way of doing this would be the standard method of centering by computers using the LEN(string) function. That is, one half the length of the item being centered could be subtracted from the central dot-column position of each table column, and this value could be used in a positioning command (such as the Tandy 27 + 16 + N1 + N2 sequence) to start the printing of the first character in the item the proper distance to the left of the column's center. Since each character is six dots wide, and we want the distance to be in number of dot columns rather than number of characters, the centering formula is $Z = (\text{center of column}) - .5 * 6 * \text{LEN}(\text{string})$. Using CC for the dot-column position of the column's center, this simplifies to $Z = \text{CC} - 3 * \text{LEN}(\text{string})$.

This procedure of centering material with respect to the boundaries of columns can be carried a good ways further. In fact, another whole program, called CENTABLE (program 11-5), is based on this idea and applies it to all the material in a column as well as the headings at the top of a column. Many times we have no need to worry about decimal points lining up, because the material is all verbal (names of people, companies, etc.) or if there is numerical information, it is continually intermixed with verbal items. If everything in each column is centered, the overall appearance of the table will usually be nicer than when each item is left-justified, that is, when each item starts at the same point in the left-hand part of the column.

PROGRAM 11-5. CENTABLE. Table with items in columns centered.

```

10 'PROGRAM 11-5: "CENTABLE" -- COLUMN-CENTERING TABLE
12 'Copyright (c) 1985 by John Warner Davenport
15 'FOR TRS-80 MODEL III OR 4 AND DMP-400 PRINTER
20 CLEAR 1000
30 CR$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)      'CAR. RET. W/O LINE FEED
40 P0$=CHR$(27)+CHR$(16)+CHR$(0)                'DOT-COLUMNS 0-255
50 P1$=CHR$(27)+CHR$(16)+CHR$(1)                ' " " 256-511
60 P2$=CHR$(27)+CHR$(16)+CHR$(2)                ' " " 512-767
70 P3$=CHR$(27)+CHR$(16)+CHR$(3)                ' " " 768-1023
80 READ T$,DY$,DT$,WK$      'READ TITLE, "DAY," "DATE," "WEEK" HEADINGS
90 TB=75-LEN(T$)
100 LPRINT CHR$(30);TAB(TB);CHR$(27);CHR$(23);CHR$(15);CHR$(27);CHR$(14);
110 LPRINT T$;CHR$(27);CHR$(15);CHR$(14);
120 FOR SPACE=1 TO 4: GOSUB 9999: NEXT SPACE
130 GOSUB 4000
140 GOSUB 1000: GOSUB 9999: GOSUB 1000: GOSUB 9999: GOSUB 1000
150 READ A$,B$,C$,D$,E$,F$,G$,H$      'READ DAYS OF WEEK
160 GOSUB 8010
170 LPRINT CR$;CHR$(18);CHR$(255);CHR$(30);DY$;: GOSUB 1000
180 GOSUB 7000
190 GOSUB 9999: GOSUB 1000: GOSUB 9999: GOSUB 1000: GOSUB 5000
200 GOSUB 1000: GOSUB 9999: GOSUB 1000: GOSUB 9999
210 WK=1
220 READ A$,B$,C$,D$,E$,F$,G$,H$      'READ DATES OF MONTH
230 GOSUB 8000
240 LPRINT CHR$(18);CR$;CHR$(255);CHR$(30);DT$;
250 GOSUB 7000
260 GOSUB 1000: GOSUB 9999: GOSUB 1000: GOSUB 9999: GOSUB 1000
270 GOSUB 6000: GOSUB 9999: GOSUB 1000: GOSUB 9999: GOSUB 1000
280 READ A$,B$,C$,D$,E$,F$,G$,H$      'READ WEATHER DESCR'N
290 GOSUB 8000: GOSUB 7000: GOSUB 1000: GOSUB 9999
300 LPRINT CHR$(18);CR$;CHR$(255);CHR$(30);WK$;WK;
310 GOSUB 1000: GOSUB 9999
320 READ A$,B$,C$,D$,E$,F$,G$,H$      'READ LOW-HIGH TEMP.
330 GOSUB 1000: GOSUB 8000: GOSUB 7000: GOSUB 1000: GOSUB 9999
340 GOSUB 1000: GOSUB 9999: GOSUB 3000: GOSUB 1000
350 IF WK=5 THEN GOSUB 1000: GOSUB 5000: GOTO 380
360 GOSUB 9999: GOSUB 1000: GOSUB 9999
370 WK=WK+1: GOTO 220
380 END
1000 '***** VERTICAL LINES *****
1010 LPRINT CHR$(18);CR$;CHR$(255);STRING$(47,128);CHR$(255);
1020 FOR V=1 TO 8: LPRINT STRING$(111,128);CHR$(255);: NEXT V
1025 LPRINT CR$;
1030 RETURN
2000 '***** HORIZONTAL DASHES (FULL LINE)*****
2010 LPRINT P0$;CHR$(2);STRING$(157,"-"): RETURN
3000 '***** HORIZONTAL SINGLE LINE (2 DOTS THICK) *****
3010 LPRINT CHR$(18);STRING$(250,140);STRING$(250,140);STRING$(250,140);STRING$(
195,140);: RETURN

```

```

4000 '***** HORIZONTAL DOUBLE LINE (HIGH) *****
4010 LPRINT CHR$(18);STRING$(250,133);STRING$(250,133);STRING$(250,133);STRING$(
195,133);: RETURN
5000 '***** HORIZONTAL DOUBLE LINE (LOW) *****
5010 LPRINT CHR$(18);STRING$(250,208);STRING$(250,208);STRING$(250,208);STRING$(
195,208);: RETURN
6000 '***** INDENTED DASHED LINE *****
6010 LPRINT CHR$(30);P0$;CHR$(48);STRING$(131,"-");: RETURN
7000 '***** PRINTING OF COLUMN CONTENTS *****
7010 LPRINT CHR$(30);
7020 IF ZA=49 THEN ZA=50
7030 LPRINT P0$;CHR$(47+ZA);ZA$;
7040 LPRINT P0$;CHR$(159+ZB);ZB$;
7050 IF ZC=81 THEN ZC=82
7060 LPRINT P1$;CHR$(15+ZC);ZC$;
7070 LPRINT P1$;CHR$(127+ZD);ZD$;
7080 IF ZE<17 THEN LPRINT P1$;CHR$(239+ZE);ZE$;: GOTO 7120
7090 IF (ZE-16)=10 OR (ZE-16)=12 THEN ZE=ZE+1 'POSSIBLE-TROUBLE-CODE DETOURS
7095 IF (ZE-16)=9 THEN (ZE-16)=8 'DETOUR OF MODEL 4'S TROUBLE CODE 9
7100 LPRINT P2$;CHR$(ZE-16);ZE$;
7110 IF ZF=1 THEN ZF=2
7120 LPRINT P2$;CHR$(95+ZF);ZF$;
7130 IF ZG<49 THEN LPRINT P2$;CHR$(207+ZG);ZG$;: GOTO 7170
7140 IF (ZG-48)=10 OR (ZG-48)=12 THEN ZG=ZG+1
7150 LPRINT P3$;CHR$(ZG-48);ZG$;
7160 IF ZH=32 THEN ZH=33
7170 LPRINT P3$;CHR$(63+ZH);ZH$;
7180 RETURN
8000 '***** POSITION AND COLUMN-STRING ASSIGNMENTS *****
8010 ZA=(116-6*LEN(A$))/2: ZA$=A$: ZB=(116-6*LEN(B$))/2: ZB$=B$
8020 ZC=(116-6*LEN(C$))/2: ZC$=C$: ZD=(116-6*LEN(D$))/2: ZD$=D$
8030 ZE=(116-6*LEN(E$))/2: ZE$=E$: ZF=(116-6*LEN(F$))/2: ZF$=F$
8040 ZG=(116-6*LEN(G$))/2: ZG$=G$: ZH=(116-6*LEN(H$))/2: ZH$=H$
8050 RETURN
9000 '***** 7/72" LINE FEED *****
9999 LPRINT CHR$(18);CHR$(13);: RETURN
10000 '***** DATA START HERE *****
10010 DATA "DAILY WEATHER LOG -- ROCHESTER, MINNESOTA (1994)"," DAY"," DATE","
WEEK"
10020 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,WEEKLY SUMMA
RY
10030 DATA APRIL 1,APRIL 2,APRIL 3,APRIL 4,APRIL 5,APRIL 6,APRIL 7," "
10040 DATA HEAVY OVERCAST,"CLOUDY, LIGHT RAIN",CLEARING,"PARTLY SUNNY,WINDY",FAI
R,PARTLY CLOUDY,RAIN,BRIEF SIGN
10050 DATA 29-42,36-45,32-44,38-55,41-59,37-50,40-46,OF SPRING
10060 DATA APRIL 8,APRIL 9,APRIL 10,APRIL 11,APRIL 12,APRIL 13,APRIL 14," "
10070 DATA RAIN/CLEARING,MOSTLY SUNNY,BEAUTIFUL!,PARTLY CLOUDY,PARTLY CLOUDY,FRE
EZING RAIN,LIGHT MIST,AN UP-AND-
10080 DATA 38-48,42-56,45-65,42-58,35-44,29-34,33-39,DOWN WEEK
10090 DATA APRIL 15,APRIL 16,APRIL 17,APRIL 18,APRIL 19,APRIL 20,APRIL 21," "
10100 DATA "CLOUDY, WINDY",PARTLY SUNNY,FAIR AND PLEASANT,SUNNY AND WARMER,JUST
LIKE SUMMER!,PARTLY SUNNY,FAIR,SPRING HAS
10110 DATA 37-51,44-60,47-63,49-68,61-79,52-68,57-70,ARRIVED!

```

10120 DATA APRIL 22,APRIL 23,APRIL 24,APRIL 25,APRIL 26,APRIL 27,APRIL 28," "

10130 DATA CLOUDING UP,"GRAY, OVERCAST",SNOW -- 9 INCHES!,SNOW/CLEARING,SUNNY AND COOL,RAIN AND SLEET,CLOUDY AND WINDY,HORRORS! BACK

10140 DATA 45-57,38-42,30-34,27-38,33-41,32-39,35-43,TO WINTER!

10150 DATA APRIL 29,APRIL 30,MAY 1,MAY 2,MAY 3,MAY 4,MAY 5," "

10160 DATA PARTLY SUNNY,"SUNNY, NICE",CLOUDY/RAIN,MORE RAIN,MORE RAIN!,CLEARING, SUNNY,APRIL SHOWERS

10170 DATA 38-57,42-64,47-59,50-58,52-62,44-69,51-72,IN MAY

10180 END,2,3,4,5,6,7,8

Figure 11-8 has an example of a column-centered table with nine columns. It takes a fifteen-inch printer to fit in so many columns, as well as a limitation of eighteen characters (including spaces) per item to stay within the boundaries of the eight centered columns. This weather summary illustrates a mixture of verbal and numerical material, with the verbal items varying widely in their length. You'll find another example back in chapter 2: Table 2-2, which presented control codes for the three families of printers, was produced by a four-column version of the CENTABLE program.

You might be wondering: Since word processors are so good at centering, why not use one of those instead of a BASIC program like CENTABLE? The answer is that most

FIGURE 11-8. A column-centered table (CENTABLE program).

DAILY WEATHER LOG -- ROCHESTER, MINNESOTA (1974)

DAY	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	WEEKLY SUMMARY
DATE	APRIL 1	APRIL 2	APRIL 3	APRIL 4	APRIL 5	APRIL 6	APRIL 7	
WEEK 1	HEAVY OVERCAST 29-42	CLOUDY, LIGHT RAIN 36-45	CLEARING 32-44	PARTLY SUNNY,WINDY 38-55	FAIR 41-59	PARTLY CLOUDY 37-50	RAIN 40-46	BRIEF SIGN OF SPRING
DATE	APRIL 8	APRIL 9	APRIL 10	APRIL 11	APRIL 12	APRIL 13	APRIL 14	
WEEK 2	RAIN/CLEARING 38-48	MOSTLY SUNNY 42-56	BEAUTIFUL! 45-65	PARTLY CLOUDY 42-58	PARTLY CLOUDY 35-44	FREEZING RAIN 29-34	LIGHT MIST 33-39	AN UP-AND- DOWN WEEK
DATE	APRIL 15	APRIL 16	APRIL 17	APRIL 18	APRIL 19	APRIL 20	APRIL 21	
WEEK 3	CLOUDY, WINDY 37-51	PARTLY SUNNY 44-60	FAIR AND PLEASANT 47-63	SUNNY AND WARMER 49-68	JUST LIKE SUMMER! 61-79	PARTLY SUNNY 52-68	FAIR 57-70	SPRING HAS ARRIVED!
DATE	APRIL 22	APRIL 23	APRIL 24	APRIL 25	APRIL 26	APRIL 27	APRIL 28	
WEEK 4	CLOUDING UP 45-57	GRAY, OVERCAST 38-42	SNOW -- 9 INCHES! 30-34	SNOW/CLEARING 27-38	SUNNY AND COOL 33-41	RAIN AND SLEET 32-39	CLOUDY AND WINDY 35-43	HORRORS! BACK TO WINTER!
DATE	APRIL 29	APRIL 30	MAY 1	MAY 2	MAY 3	MAY 4	MAY 5	
WEEK 5	PARTLY SUNNY 38-57	SUNNY, NICE 42-64	CLOUDY/RAIN 47-59	MORE RAIN 50-58	MORE RAIN! 52-62	CLEARING 44-69	SUNNY 51-72	APRIL SHOWERS IN MAY

word-processing software is designed to center material only with respect to left and right margins, and even when the software is fancy enough to provide for columns, codes for printing vertical and horizontal lines (other than underlining) aren't available.

As listed in program 11-5, the CENTABLE printout is restricted to a format of one left-justified column followed by eight item-centering columns. If several more variables were introduced, however, the number and width of columns could be determined by answers to INPUT questions early in the program (or specified in additional DATA items). This would require changes in the lines involving the positioning of vertical lines and column centers (subroutines 1000, 7000, and 8000) and the length of horizontal lines (subroutine 2000).

Table-and-graph combinations. It's also interesting to try combinations of graphs and tables, or introducing graphic features into tables. For example, you can have a conventional table that is packed with information lying to the left of a horizontal bar graph that highlights the most important statistic in the table or a derived measure based on the data in the table. With built-in characters in the table aligned with the bars, so that a row in the table is printed in the same printhead pass as the bar, the programming of such a table-graph would be fairly simple. It's also not too difficult to put a vertical bar or line graph together with a table, so that data points in the graph line up with the columns (instead of rows) of the table.

Other combinations may be more appropriate for certain situations. A table could be placed between two graphs or in the center column of a two-way horizontal graph. In either of these cases, only minor changes in a program for a two-way bar graph (such as our TWOWAYBG, program 8-5) would be needed. For that matter, you could put a single (one-way or two-way) graph between two tables.

Graphic inserts. Figure 11-9 exemplifies an entirely different approach in making tables more graphic—using bit-image constructions that are embedded within a table. In this example a homemade upright alphabet printed white-on-

1984 DETROIT TIGERS -- DAILY GAME LOG

68	DATE	WON-LS	POS	GAME SCORE	HITTING LEADERS	PITCHERS
1	4/3	1-0	1,.....	TIGERS 8, @Minnesota 1	Evans HR,3BI; Parrish 2BI; Johnson 2H	Morris (1-0)
2	4/5	2-0	1,.....	TIGERS 7, @Minnesota 3	Gibson HR,3BI; Trammell HR,4H	Petry (1-0)
3	4/6	3-0	1,.....	TIGERS 3, @Chicago 2	Bergman 2BI, Evans GMBI	Wilcox(1-0), Hernandez S1
4	4/7	4-0	1, +.5	TIGERS 4, @Chicago 0	Lemon HR, 2BI	MORRIS (2-0) NO-HITTER!
5	4/8	5-0	1, +1.5	TIGERS 7, @Chicago 3	Garbey 2H,3BI; Gibson HR,2H	Lopez (1-0)
6	4/10	6-0	1, +2.5	TIGERS 5, Rangers 1	Evans HR,3BI	Petry (2-0),4-hitter
7	4/12	7-0	1, +3	TIGERS 9, Rangers 4	Lemon HR,3BI;Trammell HR,2H;Whitaker HR	Morris (3-0)
8	4/13	8-0	1, +3	TIGERS 13, @Boston 9	Parrish HR,2BI; Lemon,Herndon 2H	Bair (1-0)
9	4/18	9-0	1, +3	TIGERS 4, Royals 3	Parrish HR,3H; Lemon,Brockens 2H	Hernandez (1-0)
10	4/19	9-1	1, +1.5	Royals 5, Tigers 2	Gibson, Evans 2H	Petry (2-1)
11	4/20	10-1	1, +2.5	TIGERS 3, White Sox 2	Parrish 3H; Garbey, Gibson 2H	Lopez (2-0)
12	4/21	11-1	1, +3.5	TIGERS 4, White Sox 1	Whitaker HR,2H; Evans,Trammell 2H	Rozema (1-0)
13	4/22	12-1	1, +4.5	TIGERS 9, White Sox 1	Evans 3H,2BI; Garbey 3BI; Lemon 4H	Berenguer (1-0)
14	4/24	13-1	1, +5	TIGERS 6, Twins 5	Trammell 2H,2BI; Whitaker 2H; Lemon HR	Morris (4-0)
15	4/24	14-1	1, +5.5	TIGERS 4, Twins 3	Parrish GMBR,3BI; Garbey 2H	Abbott (1-0), Lopez S1
16	4/25	15-1	1, +5.5	TIGERS 9, @Texas 4	Parrish HR,3BI; Whitaker 3H	Wilcox (2-0), Hern'dez S2
17	4/26	16-1	1, +6	TIGERS 7, @Texas 5	Trammell 3H; Garbey, Lemon 2H	Bair (2-0), Lopez S2
18	4/27	16-2	1, +5	Indians 8, Tigers 4 (19 innings)	Whitaker 3H,Trammell 2H (Hargrove 6SHR)	Abbott (1-1)
19	4/28	17-2	1, +5.5	TIGERS 6, Indians 2	Whitaker HR,2H,2BI; Lemon HR,2H	Morris (5-0), 3-hitter
20	4/29	18-2	1, +6	TIGERS 6, Indians 1	Gibson 3H,3BI; Whitaker 2H	Petry (3-1), 1 hit in 8
21	5/1	19-2	1, +6.5	TIGERS 11, Red Sox 2	Lemon HR(6),3H,4BI; Garbey 3H,4BI	Wilcox (3-0)
22	5/2	19-3	1, +6.5	Red Sox 5,Tigers 4	Gibson 4H, Bergman 3H	Berenguer (1-1)
23	5/3	19-4	1, +5	Red Sox 1, Tigers 0	Parrish 2H (6-hitter by Ojeda)	Morris (5-1), 5-hitter
24	5/4	20-4	1, +5	TIGERS 9, @Cleveland 2	Whitaker 4H; Herndon 3H,2BI	Petry (4-1), Hern'dez S3
25	5/5	21-4	1, +5	TIGERS 6, @Cleveland 5	Lemon HR, 4H, 3BI	Abbott (2-1), Lopez S3
26	5/6	22-4	1, +5	TIGERS 6, @Cleveland 5 (12)	Grubb HR,2BI; Trammell, Bergman 3H	Lopez (3-0)
27	5/7	23-4	1, +5.5	TIGERS 10, @Kansas City 3	Trammell 3H; Lemon 2H,3BI; Grubb HR,2BI	Berenguer (2-1), Bair S2
28	5/8	24-4	1, +6	TIGERS 5, @Kansas City 2	Trammell 6SHR; Lemon 2H, 2BI	Morris (6-1), 7-hitter
29	5/9	25-4	1, +7	TIGERS 3, @Kansas City 1	Trammell, Evans 3H; Kuntz, Whitaker 2H	Petry (5-1), Lopez S4
30	5/11	26-4	1, +7.5	TIGERS 8, Angels 2	Bergman 2H,3BI; Gibson HR,2H; Evans 2BI	Wilcox (4-0), Hern'dez S4
31	5/12	26-5	1, +7.5	Angels 4, Tigers 2	Lemon, Whitaker 2H	Berenguer (2-2)
32	5/14	27-5	1, +8	TIGERS 7, Mariners 5	Trammell HR,3H,2BI; Kuntz HR,3H	Lopez (4-0)
33	5/15	28-5	1, +8	TIGERS 6, Mariners 4	Johnson 2BI, Whitaker 2H	Morris (7-1), Hern'dez S5
34	5/16	29-5	1, +8	TIGERS 10, Mariners 1	Kuntz, Castillo 2BI; Whit'r,Parrish 2H	Wilcox (5-0)
35	5/18	30-5	1, +7.5	TIGERS 8, Athletics 4	Parrish 2H,2BI; Gibson 2BI; Garbey 2H	Petry (6-1), 7-hitter
36	5/19	31-5	1, +7.5	TIGERS 5, Athletics 4	Evans 3H,2BI; Whitaker HR,2BI	Morris (8-1), Lopez S5
37	5/20	32-5	1, +8	TIGERS 4, Athletics 3	Herndon 2H, Lowry HR	Wilcox (6-0), Hern'dez S6
38	5/22	33-5	1, +8	TIGERS 3, @California 1	Whitaker 2H	Berenguer (3-2), Lopez S6
39	5/23	34-5	1, +8	TIGERS 4, @California 2	Parrish HR,3H,2BI;Castillo,Trammell 2H	Petry (7-1), Hern'dez S7
40	5/24	35-5	1, +8.5	TIGERS 5, @California 1	Trammell HR,2BI; Parrish HR,2H	Morris (9-1), 4-hitter
41	5/25	35-6	1, +7.5	@Seattle 7, Tigers 3	Trammell, Evans 2H; Johnson 2BI	Wilcox (6-1)
42	5/26	35-7	1, +6.5	@Seattle 9, Tigers 5	Gibson, Kuntz HR,2H	Berenguer (3-3)
43	5/27	35-8	1, +5	@Seattle 5, Tigers 1	Trammell 3H, Parrish 2H	Petry (7-2)

FIGURE 11-9. Bit-image graphic emphasis in a table (GAMELOG program).

black is inserted into many lines of a table to emphasize trends, winning streaks, or other features. Program 11-6, called GAMELOG, shows how the regular characters of the table are intermixed, within lines, with the bit-image highlighting.

PROGRAM 11-6. GAMELOG. Graphic-emphasized table.

```

10 'PROGRAM 11-6: "GAMELOG" (1984 DETROIT TIGERS' 1ST 43 GAMES)
15 'Copyright (c) 1985 by John Warner Davenport
20 'FOR TRS-80 MODEL III OR 4 AND 8- OR 15-INCH TRS-80 PRINTERS
30 CLEAR 500
40 P$=CHR$(27)+CHR$(16)                                ' POSITIONING PREFIX
50 P0$=CHR$(27)+CHR$(16)+CHR$(0)+CHR$(0)                ' CAR. RET. W/O LINE FEED
60 CD$=CHR$(30)+CHR$(27)+CHR$(20)+CHR$(18)              ' CONDENSED PRINTING
70 NM$=CHR$(30)+CHR$(27)+CHR$(19)+CHR$(18)              ' NORMAL DENSITY
80 LPRINT CHR$(30);CHR$(27);CHR$(32);                   ' (BOLD OFF FOR ELONGATED TITLE)
90 FOR N=1 TO 1: LPRINT CHR$(30);CHR$(15);CHR$(27);CHR$(16);CHR$(0);CHR$(60);CHR$
  $(27);CHR$(14);CHR$(27);CHR$(17);"1984 DETROIT TIGERS -- DAILY GAME LOG";CHR$
  (27);CHR$(19);CHR$(14);CHR$(27);CHR$(15);: NEXT N
100 LPRINT STRING$(2,13);                                ' TWO LINE FEEDS
120 LPRINT CHR$(30);CHR$(27);CHR$(20);" G#  DATE WON-LST  POS";CHR$(27);CHR$(19)
  ;"      GAME SCORE      ";CHR$(27);CHR$(16);CHR$(1);CHR$(5);" HITTING L
  EADERS";CHR$(27);CHR$(16);CHR$(1);CHR$(145);"PITCHERS";CHR$(13);
130 LPRINT CHR$(18);CHR$(28);CHR$(255);CHR$(133);CHR$(28);CHR$(226);CHR$(133);
140 GOSUB 280
150 READ G$,SC$,H$,M$,HA$: IF G$="END" THEN GOTO 1010
160 LPRINT CHR$(30);CHR$(27);CHR$(19);P$;CHR$(1);CHR$(223);CHR$(18);STRING$(2,25
  5);CHR$(30);P0$;CHR$(27);CHR$(20);CHR$(18);STRING$(2,255);CHR$(30);G$;CHR$(2
  7);CHR$(19);
170 IF LEFT$(SC$,6)="TIGERS" THEN GOSUB 1020: GOTO 200
180 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(95);CHR$(18);CHR$(255);STRING$(6,128);
  CHR$(30);CHR$(27);CHR$(17);SC$;CHR$(27);CHR$(19);: GOTO 210
190 '
200 LPRINT CHR$(30);CHR$(27);CHR$(17);MID$(SC$,8);
210 LPRINT CHR$(27);CHR$(19);CHR$(27);CHR$(16);CHR$(0);CHR$(236);CHR$(27);CHR$(2
  0);CHR$(27);CHR$(17);CHR$(18);CHR$(255);CHR$(30);HE$;
220 LPRINT CHR$(27);CHR$(19);CHR$(27);CHR$(16);CHR$(0);CHR$(240);CHR$(27);CHR$(2
  0);H$;
230 LPRINT CHR$(27);CHR$(19);CHR$(27);CHR$(16);CHR$(1);CHR$(127);CHR$(27);CHR$(2
  0);CHR$(18);CHR$(255);CHR$(30);CHR$(32);M$;
240 LPRINT CHR$(18);CHR$(13);
250 GOSUB 280
260 GOTO 150
270 '***** SUBROUTINE FOR PRINTING VERTICAL DIVIDERS *****
280 LPRINT P0$;CD$;STRING$(2,255);NM$;P$;CHR$(0);CHR$(95);NM$;CHR$(255);P$;CHR$(
  0);CHR$(236);CD$;CHR$(255);NM$;P$;CHR$(1);CHR$(127);CD$;CHR$(255);NM$;P$;CHR
  $(1);CHR$(223);STRING$(2,255);
290 IF HA$="*" THEN LPRINT P0$;STRING$(255,136);STRING$(226,136);
300 LPRINT CHR$(13);
310 RETURN
500 '***** DATA FORMAT *****

```

```

510 ' G$="GAME # DATE WON-LOST LEAD"
520 ' SC$="SCORE OF GAME (TIGERS, Opponent OR Opponent, Tigers)"
530 ' H$="HITTING LEADERS (HITS, HOME RUNS, RUNS BATTED IN)"
540 ' M$="PITCHERS (W, L, SAVE)"
550 ' HA$="END OF HOME STAY OR END OF ROAD TRIP (0, *)
560 DATA " 1 4/3 1-0 1,.....","TIGERS 8, @Minnesota 1","Evans HR,3BI; Parr
ish 2BI; Johnson 2H","Morris (1-0)",0
570 DATA " 2 4/5 2-0 1,.....","TIGERS 7, @Minnesota 3","Gibson HR,3BI; Tra
mmell HR,4H","Petry (1-0)",0
580 DATA " 3 4/6 3-0 1,.....","TIGERS 3, @Chicago 2","Bergman 2BI, Evans G
WBI","Wilcox(1-0), Hernandez S1",0
590 DATA " 4 4/7 4-0 1, +.5","TIGERS 4, @Chicago 0","Lemon HR, 2BI ","MOR
RIS (2-0) NO-HITTER!",0
600 DATA " 5 4/8 5-0 1, +1.5","TIGERS 7, @Chicago 3","Garbey 2H,3BI; Gibso
n HR,2H","Lopez (1-0)",*
610 DATA " 6 4/10 6-0 1, +2.5","TIGERS 5, Rangers 1","Evans HR,3BI","Petry
(2-0),4-hitter",0
620 DATA " 7 4/12 7-0 1, +3 ","TIGERS 9, Rangers 4","Lemon HR,3BI;Trammell
HR,2H;Whitaker HR","Morris (3-0)",*
630 DATA " 8 4/13 8-0 1, +3 ","TIGERS 13, @Boston 9","Parrish HR,2BI; Lemo
n,Herndon 2H","Bair (1-0)",*
640 DATA " 9 4/18 9-0 1, +3 ","TIGERS 4, Royals 3","Parrish HR,3H; Lemon,B
rookens 2H","Hernandez (1-0)",0
650 DATA " 10 4/19 9-1 1, +1.5","Royals 5, Tigers 2","Gibson, Evans 2H","Pet
ry (2-1)",0
660 DATA " 11 4/20 10-1 1, +2.5","TIGERS 3, White Sox 2","Parrish 3H; Garbey,
Gibson 2H","Lopez (2-0)",0
670 DATA " 12 4/21 11-1 1, +3.5","TIGERS 4, White Sox 1","Whitaker HR,2H; Eva
ns,Trammell 2H","Rozema (1-0)",0
680 DATA " 13 4/22 12-1 1, +4.5","TIGERS 9, White Sox 1","Evans 3H,2BI; Garbe
y 3BI; Lemon 4H","Berenguer (1-0)",0
.
.
.
950 DATA " 40 5/24 35-5 1, +8.5","TIGERS 5, @California 1","Trammell HR,2BI;
Parrish HR,2H","Morris (9-1), 4-hitter",0
960 DATA " 41 5/25 35-6 1, +7.5","@Seattle 7, Tigers 3","Trammell, Evans 2H;
Johnson 2BI","Wilcox (6-1)",0
970 DATA " 42 5/26 35-7 1, +6.5","@Seattle 9, Tigers 5","Gibson, Kuntz HR,2H"
,"Berenguer (3-3)",0
980 DATA " 43 5/27 35-8 1, +5 ","@Seattle 5, Tigers 1","Trammell 3H, Parrish
2H","Petry (7-2)",*
1000 DATA END,"","",""
1010 PRINT "JOB DONE": END
1020 REM -- SUBROUTINE FOR WHITE-ON-BLACK "TIGERS"
1030 LPRINT CHR$(30);CHR$(27);CHR$(19);
1040 R$=STRING$(1,255)+STRING$(2,128)+CHR$(246)+CHR$(230)+CHR$(198)+CHR$(144)+CH
R$(185)+CHR$(255)
1050 E$=CHR$(255)+STRING$(2,128)+STRING$(2,182)+STRING$(2,190)+CHR$(156)+CHR$(25
5)
1070 S$=STRING$(1,255)+CHR$(217)+CHR$(144)+STRING$(3,182)+CHR$(132)+CHR$(205)+CH
R$(255)
1090 T$=CHR$(255)+CHR$(252)+STRING$(2,254)+STRING$(2,128)+STRING$(2,254)+CHR$(25
2)+CHR$(255)

```



```

1100 I$=CHR$(255)+CHR$(190)+STRING$(2,128)+CHR$(190)+CHR$(255)
1110 G$=CHR$(255)+CHR$(193)+CHR$(128)+CHR$(190)+CHR$(182)+CHR$(150)+CHR$(132)+CHR$(133)+CHR$(255)
1120 LPRINT CHR$(27);CHR$(16);CHR$(0);CHR$(95);CHR$(18);CHR$(255);STRING$(3,128);CHR$(30);CHR$(27);CHR$(23);CHR$(18);STRING$(3,255);T$;I$;G$;E$;R$;S$;STRING$(3,255);STRING$(3,128);
1130 LPRINT CHR$(30);CHR$(8)+CHR$(122);CHR$(20);CHR$(27);CHR$(30);CHR$(19);
1140 LPRINT CHR$(27);CHR$(51);CHR$(27);CHR$(51);
1150 LPRINT CHR$(18);STRING$(58,152);
1160 LPRINT CHR$(30);CHR$(8);CHR$(116);
1165 FOR N=1 TO 34: LPRINT CHR$(27);CHR$(51);: NEXT N
1170 LPRINT CHR$(18);STRING$(58,134);STRING$(2,128);
1180 LPRINT CHR$(30);CHR$(20);CHR$(27);CHR$(30);CHR$(19);
1190 RETURN

```

The all-ranks table. Programs for historical point-and-range graphs, such as PNTRANGE (program 9-3), can be changed to make tables that are interesting and very informative. Instead of displaying the absolute values of one highlighted team against the best and worst teams of all teams in, say, won-lost percentage, we can convert these values to ranks. We can also use one-letter abbreviations for all the unhighlighted teams to show their final positions in the standings every year in columns of a table. Finally, the rank of the highlighted team can be made to stand out clearly with a black-white reversal. With these features, originally suggested by Jeffrey Neuman of Macmillan Publishing Company, we would have a new (or at least little-known) kind of table.

In figure 11-10 we have an example of this kind of table. Or is it a graph? Let's call it a "grable."

This grable, for the St. Louis Cardinals, shows the trends in their successes and failures from 1901 through 1982. You can follow the Cardinals with just a glance, and with closer scrutiny you can see which teams came out better or worse than the Cardinals did in each year. You can also fix your attention on a single letter from column to column to follow one of the unhighlighted teams over the decades.

With a pair of grables lined up vertically, as in figure 11-11, you can display relationships in time-series fashion, often more clearly than with correlation scatterplots. The highlighted team in figure 11-11 is the Boston Red Sox, and as a whole the figure makes the dependence of their winning and losing (upper grable) on the quality of their pitching (lower) painfully obvious.

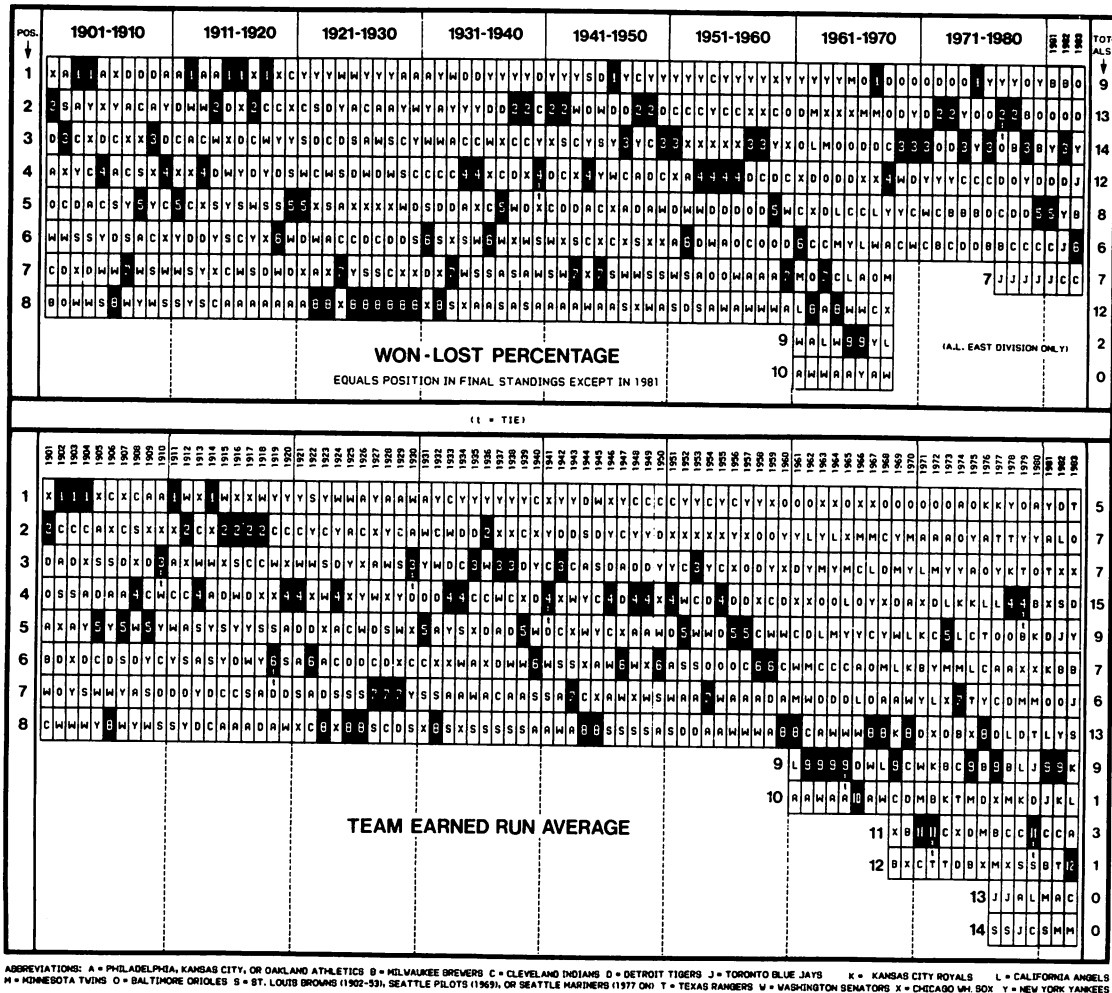


FIGURE 11-11. The relationship between winning and pitching for the Boston Red Sox over eighty-three years, shown by two vertically aligned grables.

though the GRABLE program has the years listed at the bottom of the chart, they were moved to the top of the lower grable in Figure 11-11, again by hand (and glue). Such “cheating” is the subject of the last chapter.

If it is your impression that diagrams are more fun to do than graphs on a personal computer and printer, then you have gleaned another message from this chapter.

Computer-Aided Graphics

IN the last several chapters we've seen how graphic displays can be programmed for the dot-matrix printer for a variety of often limited purposes. Now it's time to get down to business and see how the printer can help out those of us (not necessarily full-fledged professionals) who have some more ambitious projects to tackle.

By now you should have a pretty good idea of what sorts of things the computer and printer do best—and where they tend to fail—when it comes to drawing pictures, graphs, diagrams, or whatever. On the other hand, there are graphic features that can obviously be done better by hand than on the printer. But if you use only manual techniques and face the task of drawing hundreds of lines, data points, and other features in a single graphic creation, you'll wish there was some easier way to do the tedious, repetitive part of the job.

Conceivably, if you combine computerized and manual drafting, many kinds of graphic productions will turn out better than either way alone. And especially if “better” can mean “bigger” (among other qualities), the combination might produce graphic displays that are not just a little better but five or ten times better. For that matter, there are probably many projects in graphics that can't reasonably be done in the first place unless part of the work is done by a computer.

There is a nice dovetailing to notice here, too. Computers are famous for handling endlessly repetitive tasks without getting tired or bored, and for grinding out many of those

tedious jobs that can drive a graphic artist up the wall. We've also seen that the ordinary dot-matrix printer can achieve a high level of accuracy in drawing graphs and diagrams—in many cases a better job in precise positioning of data points, grid lines, boxes containing letters, etc., than we find in the usual products of the drafting table.

But the computer and printer fall far short of the graphic artist when it comes to professional finishing touches, use of pictorial material, and even some more routine things such as labeling and drawing oblique lines. Also, there's no printer sold in the microcomputer market that is wider than fifteen inches, and yet some of our daily demands in graphics call for displays that are several square feet in size; the best the printer can do in these cases is provide the components, which have to be assembled and finished by hand—but the printer can certainly provide many of those components in good quality as well as quantity.

In a nutshell: Why not use the computer-controlled printer where repetition and accuracy are demanded and use the manual techniques for final polishes and for assembling constructions of unusual size?

To appreciate the possibilities here, we need to look at some examples that go beyond the earlier ones in the book. Since the author's professional experience in this form of printer graphics has been pretty much confined to sports-graphing up to now, most of our examples will involve baseball history and statistics.

Rough Overviews

Often it happens that a graph we're trying to design is supposed to summarize a fairly huge amount of information in a simple, clear fashion. But aside from having a rough idea of the highest and lowest values in the whole set of data, we have no good notion of the overall shape of the data, much less the many smaller features in it. In situations like this, it may be helpful to go through a preliminary step that is radically in the opposite direction from simplicity and clarity—have the printer plot *all* the data.

If we already have a program such as PNTRANGE (program 9-3) close by, this will be easy to do. Since this program for point-and-range graphs has to read every data item in

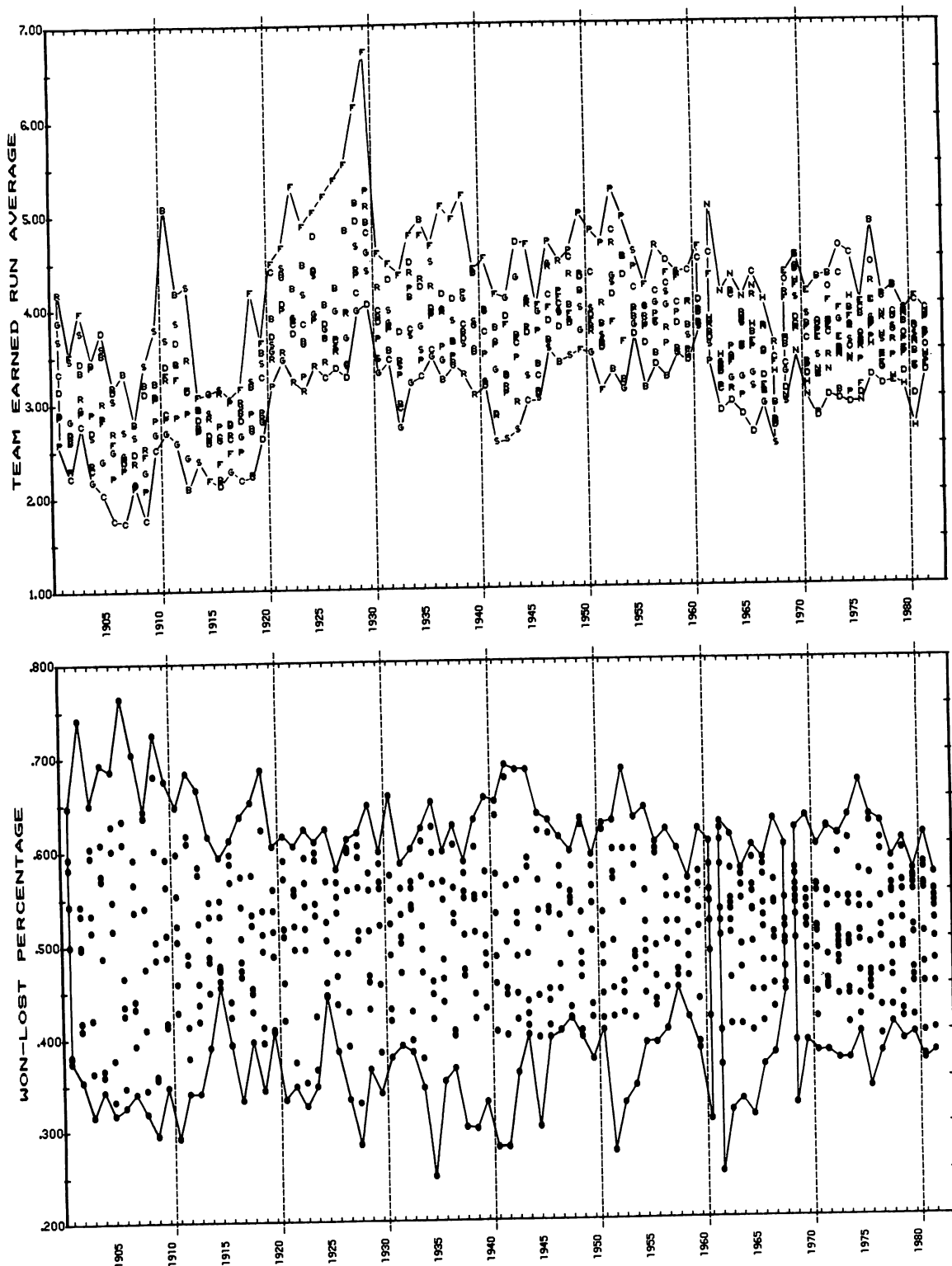


FIGURE 12-1. Plotting all the data you have, as in these two examples, may be messy, but it can give you a quick and detailed overview to use in designing simpler versions of a graph.

order to determine the best and worst performances of each year, we need only add some data-point symbols and positioning codes to make it plot every scrap of data it reads. As figure 12-1 shows, the result of doing this may be an ugly sight, but it accomplishes its purpose.

The upper graph in this figure portrays the team earned-run average for every National League team for every year in the period of 1901 to 1981. Capital letters were used as data points, and so that they wouldn't all be on top of one another, the smallest possible homemade alphabet—perpendicular five-by-five—was used for these. Inked lines were added manually to connect all the worst performances and all the best performances, and also (vertically) to show when there were changes in the number of teams in the league. These lines, which aren't absolutely necessary, help you to see the trends and general shape over the years. By paying attention to the team letters, you can spot when one team is departing dramatically from the others, such as the terrible pitching by the F team (the Philadelphia Phillies) in the 1920s and 1930s.

The lower graph (National League won-lost percentages) shows another way of programming the all-data graph—black dots instead of letters. You lose the identity of the data points this way, but that's no problem if you're mainly interested in the overall shape and how the performances in each year are distributed. Some graphs of this type may be passable as range graphs in final form.

Precision Plotting

The sheer size of the data set in these two graphs further emphasizes how surprisingly accurate a dot-matrix printer is in horizontal positioning. As with many of our graphs in this book, what is vertical in these two graphs was horizontal (parallel to the printhead movement) in the printing. Besides plotting the data points with more accuracy than is typical of manual drafting, each sweep of the printhead also prints a short segment of the top and bottom X axes. You would be wise to worry about whether a printer can draw straight lines for X axes that are at right angles to the printhead's movement this way, but even in bidirectional printing (on the

DMP-400 and LPVIII at least) they come out about as straight as with any other way of drawing a line.

Precision in vertical positioning is sometimes more of a problem, as we've seen with those tiny line feeds from $\frac{3}{216}$ inch to $\frac{5}{216}$ inch (see figure 5-4 for an example). But with most other line-feed sizes, there is impressive consistency by the printer in spacing parallel horizontal lines evenly (these would be vertical lines in most of our graphs). In fact, about the only time the consistency breaks down with normal line feeds is when the printing is near a perforation in the paper or when the start of printing is right after turning the printer's platen by hand, instead of starting where the last printout left the platen.

So most of the time you can rely on the printer to make printouts up to several feet long without having to worry about vertical or horizontal lines becoming nonparallel or unevenly spaced. This is wonderful for large panoramic graphs. Figure 12-2, which shows Boston's weather over all 365 days of 1982, was a single, continuous printout from the DMP-400 measuring 37 inches long and $14\frac{1}{2}$ inches high before it was reduced for this book. With religious dedication, the printer marched majestically through the months plotting the mean temperature in relation to the expected normal temperature for each day of the year without a single slipup in the plotting of data, axes, or grid lines. (Cutting the printout for photographic reduction and then gluing the sections back together again did put a crimp in the vertical line separating August and September, however.)

The final product was a remarkable simulation of Robert Lautzenheiser's five-column graph in the *Boston Globe* on January 1 of 1983. The computer did 98% of the work in that final product: everything but the placement of the Y-axis lettering, the MONTHLY EXTREMES label, and the months along the top of the graph. And even those labels that were pasted on were printed by the DMP-400 using the WRDCRAFT program (program 7-2) or built-in characters. The main discrepancy between the two renditions is that the computer was programmed to print both the solid precipitation bar and the vertical-line-shaded temperature bar, as well as the axes and grid lines, in single printhead passes, whereas the presentation in the *Globe* was in the form of two separate, vertically aligned bar graphs.

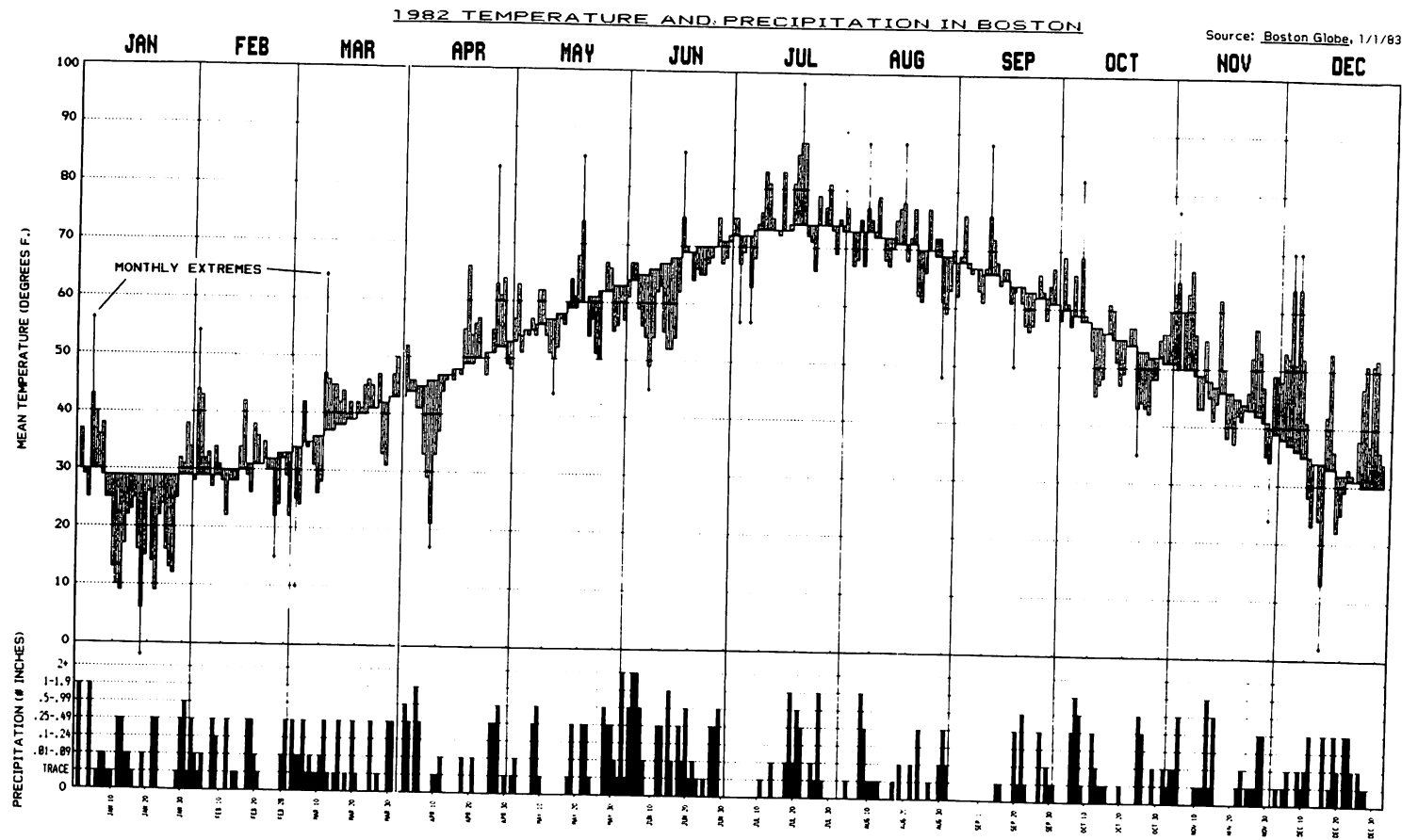


FIGURE 12-2. A panoramic weather graph.

Manual Polishing

Quite a few of the programmed graphic techniques in this book have been tested in a series of baseball team books published by Macmillan, thus far on the St. Louis Cardinals (*The Cardinals*, 1983), Chicago White Sox (*SOX: The Complete Record of the Chicago White Sox*, 1984), that beloved team of New Englanders (*Boston Red Sox*, 1984), and the world champions of 1984 (*The Detroit Tigers*, 1985). "Tested" here means submitting many printer-produced graphs to scrutiny to see whether they are publication-worthy without any finishing touches done by hand (other than a title at the top) or, failing that, how much manual polishing or assembly they needed to meet standards.

Out of roughly two dozen different displays representing over a dozen different kinds of graphs, diagrams, and tables, only one (a horizontal bar graph similar to figure 8-1) made the grade "as is." Several others came about as close as the Boston weather graph (manual labeling using the printer as a typesetting machine or dry-transfer lettering); these included a set of four graphs like those in figures 11-10 and 11-11, the tandem bar-and-line graph in figure 9-12, two tables similar to those in figures 11-5 and 11-6, and more bar graphs. Others, most commonly in the category of line graphs, needed a lot more manual finishing for the sake of clarity after large photographic reductions or just a more pleasing physical appearance.

An example from this latter group appears in figure 12-3, a two-way rank-and-range graph. This graph displays the runs scored and allowed by the Boston Red Sox over the 1901-1983 period in relation to other American League teams. It was designed to be half of a two-page spread showing home and road games separately.

The printer's part of the work on this one was considerable: the scale labels and numbers under Defense and Offense, the numbers representing all the data points of the highlighted team (even those covered up by the white-on-black numbers), the horizontal lines defining the range, the horizontal lines marking every ten years, and the rest of the numbering. The manual work included the large lettering and arrows at the top and the oblique lines connecting the data points, which involved dry-transfer symbols used in two

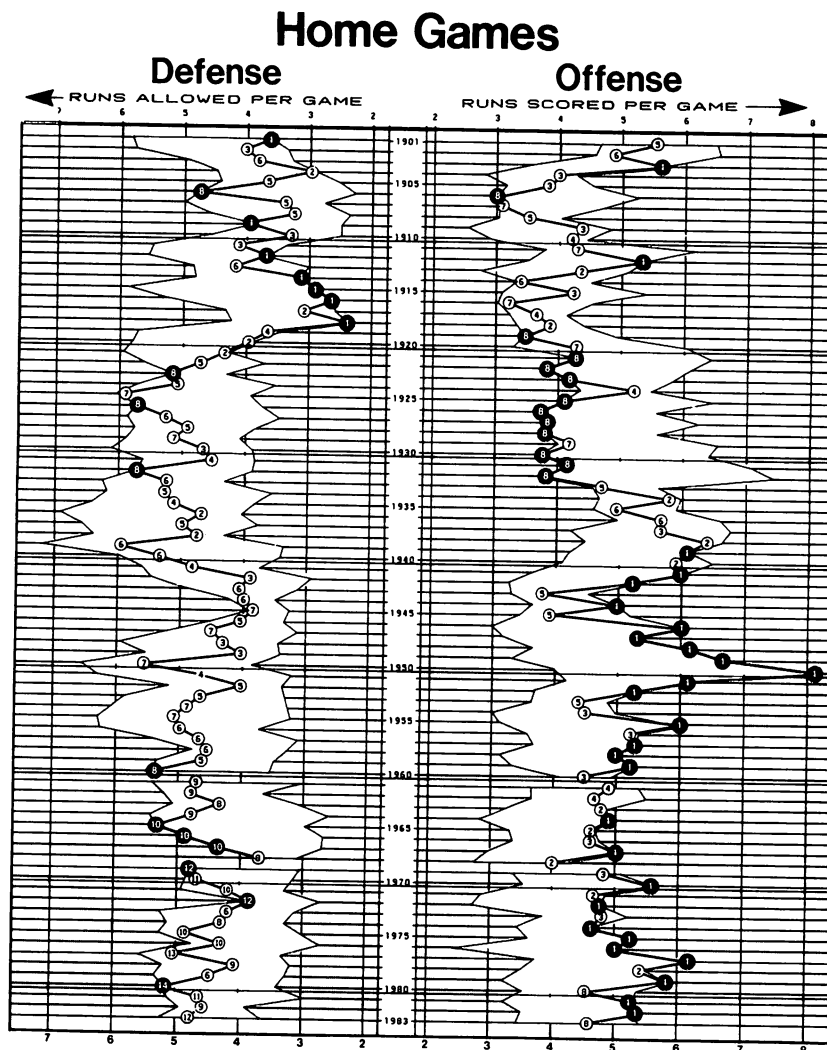


FIGURE 12-3. Dressing up a panoramic graph with manual drafting aids.

ways—every year that the Red Sox led or trailed the rest of the league, their rank was shown with white numbers on black circles, and every other year their computer-printed rank numbers were enclosed by open circles.

In a further attempt to dress up the graph, the horizontal lines defining the range were covered by light plastic shading, and the shading was cut with a graphics knife so that the jagged edges of the shading would also serve as connecting lines for the range values. It was at this point that the graph was published, rather unfortunately, because its clarity and

American League Standings

Games Won, 1977 - 84

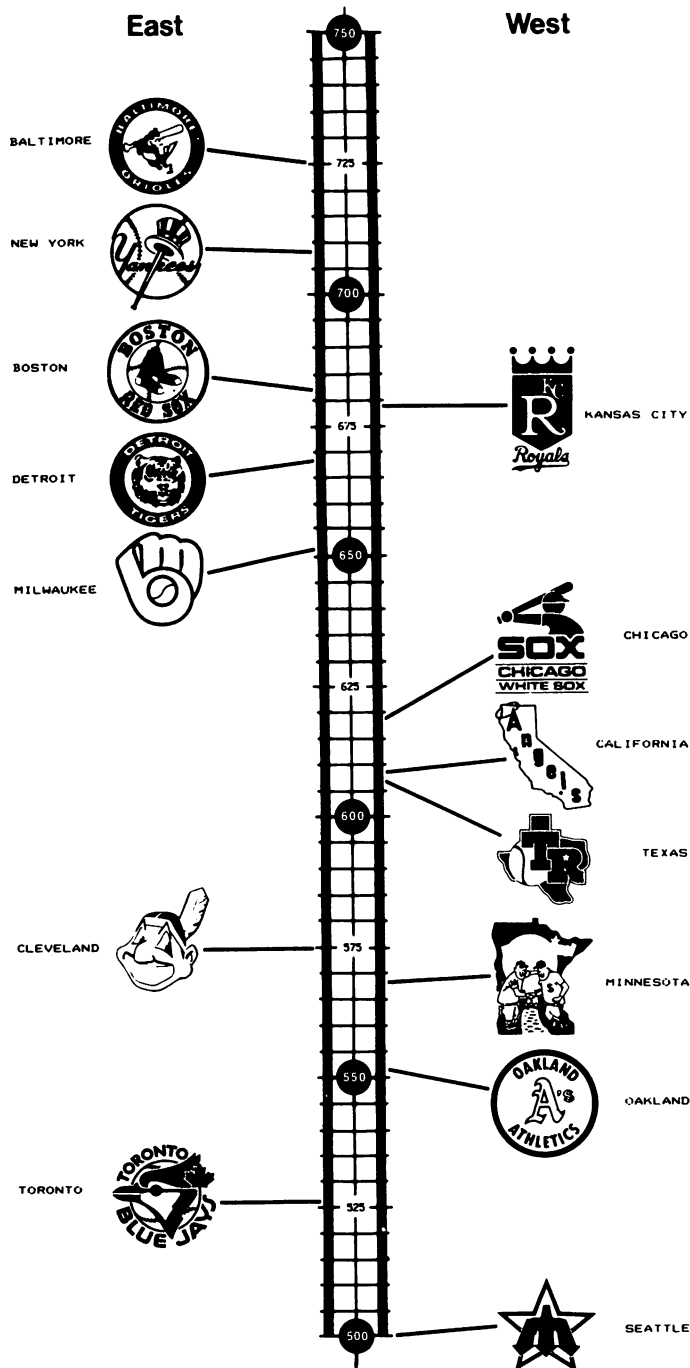


FIGURE 12-4. A ladder graph with visual elements as “data points.”

appearance were much enhanced by adding $\frac{1}{64}$ -inch solid-black Formatt tape over the jagged edges. The computer's part of the work took about ten minutes and the manual part about five hours, but without this collaboration the part done by the printer would have fallen far short of standards.

The drafting tapes, shading screens, and dry-transfer sheets can be used for hundreds of tricks in manual polishing. If you are serious about producing publication-quality graphs drawn primarily with a dot-matrix printer, you should be prepared to use some of these materials, especially with six-, seven-, or eight-pin printers. Let's not drop the standards just because something can be done by a computer.

Beyond the materials in the graphics catalogues, there are pictorial flourishes that can be introduced (manually) into computer-drawn displays, just as readily as they already are in all-manual displays. Look at almost any of the uses of photographic material for adding some razzle-dazzle in journalism, and notice how often the same sort of pictorial background scene or foreground figure could be combined with the products of the printer, especially if overlays are used. This includes using color photography in combination with computer printouts made with different colors of ink. Aside from background or foreground use of photographic material, visuals themselves can be the "data points" of a graph—see figure 12-4's ladder graph.

Multigraphs

Another whole dimension of computer-aided graphics (or manually aided printer graphics) is the assembly of many single graphs and diagrams into larger constructions we can call multigraphs. Often there's no better way to display a wide swath of history in a field, or set up comparisons without making the reader turn pages, or cover a topic in an integrated fashion.

A straightforward example of a multigraph is shown in figure 12-5. Here the computer and printer have provided us a series of individual graphs that display the entire careers of a team's best hitters. The total-bases device is used again, with segmented vertical bars showing the usual components of singles, doubles, triples, and home runs, plus additional

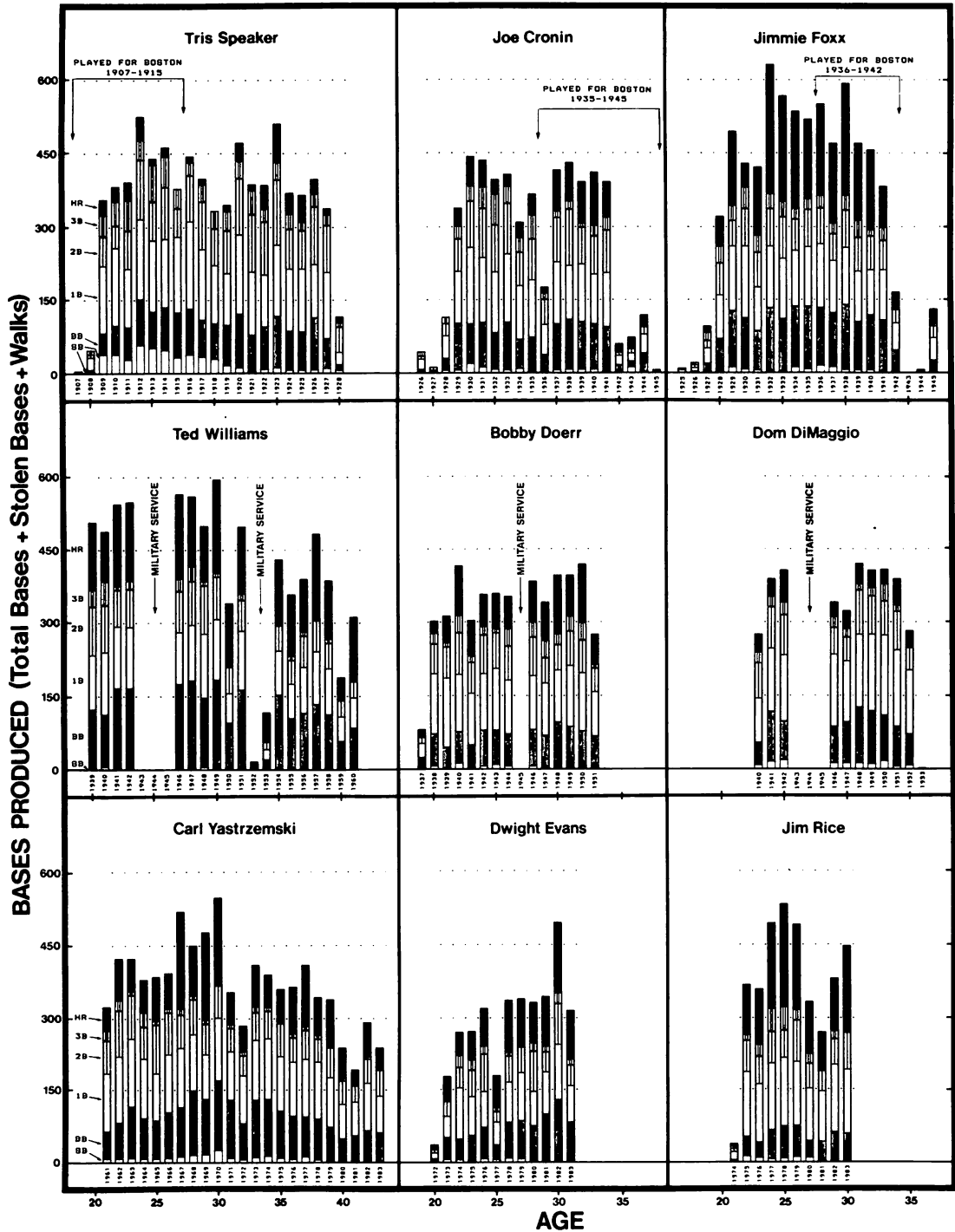


FIGURE 12-5. A whole batch of printouts can end up in a single multigraph.

bases produced by stolen bases and walks. Nine of the leading batters in Boston Red Sox history are displayed in chronological order.

You can start with full-size printouts from a fifteen-inch (or smaller) printer in assembling multigraphs of this sort. Before any pasting, a large sheet of faint-line graph paper, say, thirty by forty-eight inches, has to be marked for the locations of the nine graphs, the dividing lines between the panels, and the main axes of the multigraph. If a new X axis is used for the multigraph, as in figure 12-5, then you have to take extra care to line up the values on the X axes of the original printouts (in this case the years for each player dictated how far left or right a printout had to be placed within each of the nine panels, so that the player's age in any given year could be determined from the general age scale at the bottom). Just as much care is needed to line up the printouts' X axes with the zero points on the Y scales, which in the finished multigraph are not part of any original printout.

When the printouts are correctly positioned, they are glued or taped in place, and the rest is done with drafting tapes and dry-transfer sheets. Solid-black tapes frame and divide the set of panels, and the original X axes are covered and joined with a thin black tape. All lettering and numbering on the left and bottom sides of the multigraph is best done using rub-on characters, and some labeling inside the panels needs to be done by hand also. The amount of manual labor for a multigraph like this might come to several hours, but imagine how many more hours it would take to do the bar graphs by hand!

Sometimes the computer and printer can do most of the assembling of a multigraph and guarantee good alignment of the components in the process. An example of this is the set of World Series gamegrams in figure 12-6. The diagrams for all seven games were produced in a single continuous printout about two feet long with perfect vertical alignment. The manual finishing touches were adding inked connecting lines for the score display in the center of each diagram, applying the large game numbers over each gamegram by dry-transfer, pasting on pitchers' names, and using a combination of drafting tape, dry-transfer numbers, and photographically enlarged letters from the printer for the runs-hits-errors summary of each game. (An explanation of

the diagrams' symbols and abbreviations accompanies figure 11-6 in the previous chapter.)

Events like those over a full set of World Series games cry for integration and order when it comes to displaying them graphically. Even if it takes a little squinting at the tiny symbols or names, it's better to have such events collected in one place in chronological order than spread over several pages.

The biggest project for the Macmillan books was to present over two hundred graphs showing teams' weekly progress in pennant races from 1901 to 1980. The desired form was one showing how a single team stacked up against the range of all of its competitors from April to October within each season. After the computer had been programmed to produce reversed rank-and-range graphs like the one in figure 9-11, it wasn't too difficult (once the data were entered) for the printer to generate all the printouts needed for ten-year multigraphs and confine the eighty-year displays to eight pages. One of these decade displays, the 1911-20 period for the Chicago White Sox, is shown in figure 12-7.

This is not just another example of how multigraphs can present huge amounts of data. It is also an example of how two different computer programs can be teamed to produce the components of a multigraph. The modified version of PNTRANGE for the DMP-400 (program 9-3) plotted the Y axis (including its numbering with perpendicular seven-by-eleven characters), the White Sox' rank numbers and their connecting lines, the black range of other teams, and the grid lines in each panel. Then another program called FINALIST used the homemade nine-pin Epson FX-80 characters (up-right nine by twenty-six) to print the listings of team abbreviations (other than "White Sox") and their games-behind numbers at the right end of each panel. Printing on blank white paper, FINALIST spaced the teams according to their finishing won-lost percentages in each season, and the listings of the final standings were cut out and pasted onto the PNTRANGE printouts in the appropriate positions. Otherwise the manual finishing touches were like those for the multiple bar graph in figure 12-5.

Obviously multigraphs can be constructed in all sorts of ways, and almost any kind of graphic display can be a component of a multigraph. As figure 12-8 shows, even the unlikely pie chart can play this role. In this case, everything

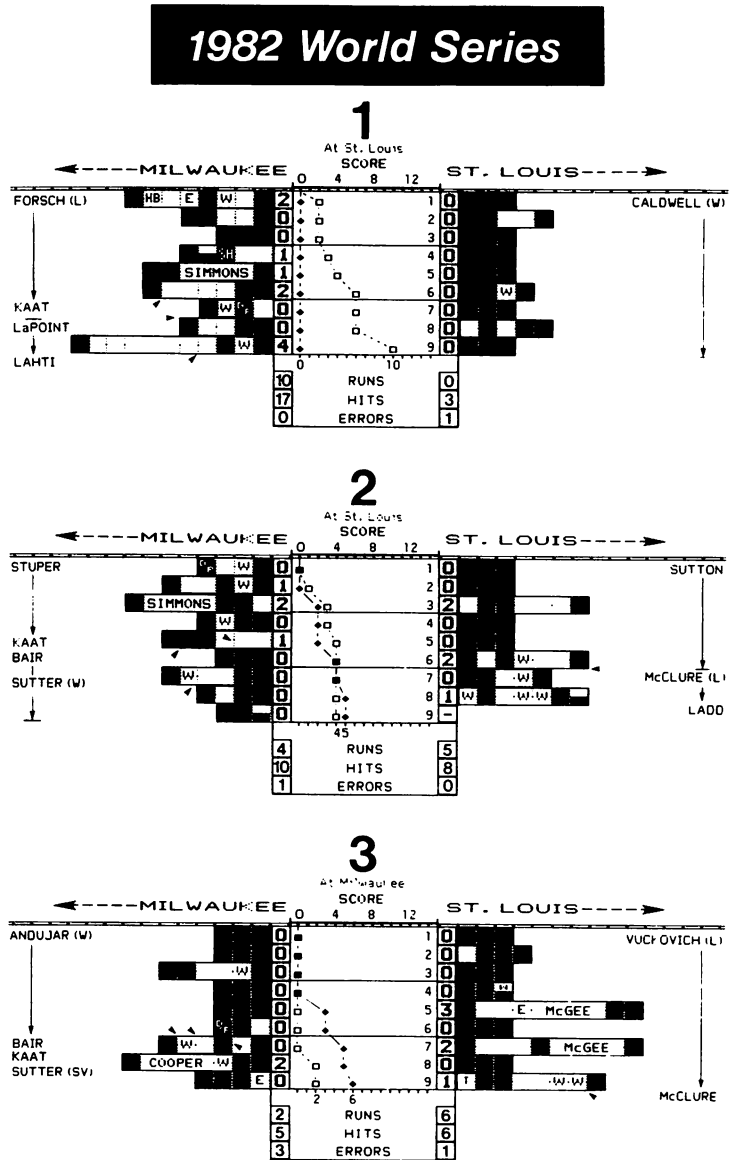
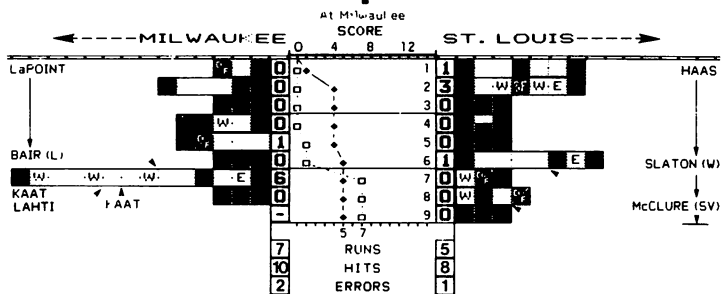
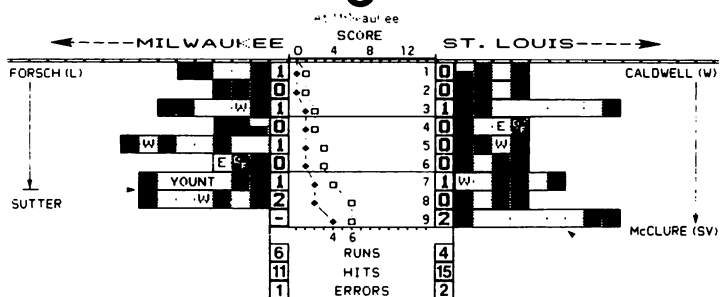


FIGURE 12-6. A multigamegram.

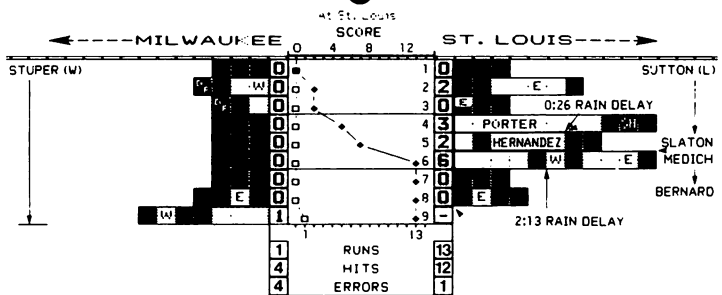
4



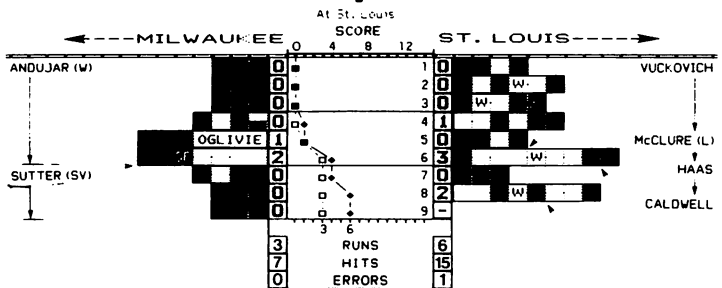
5



6



7



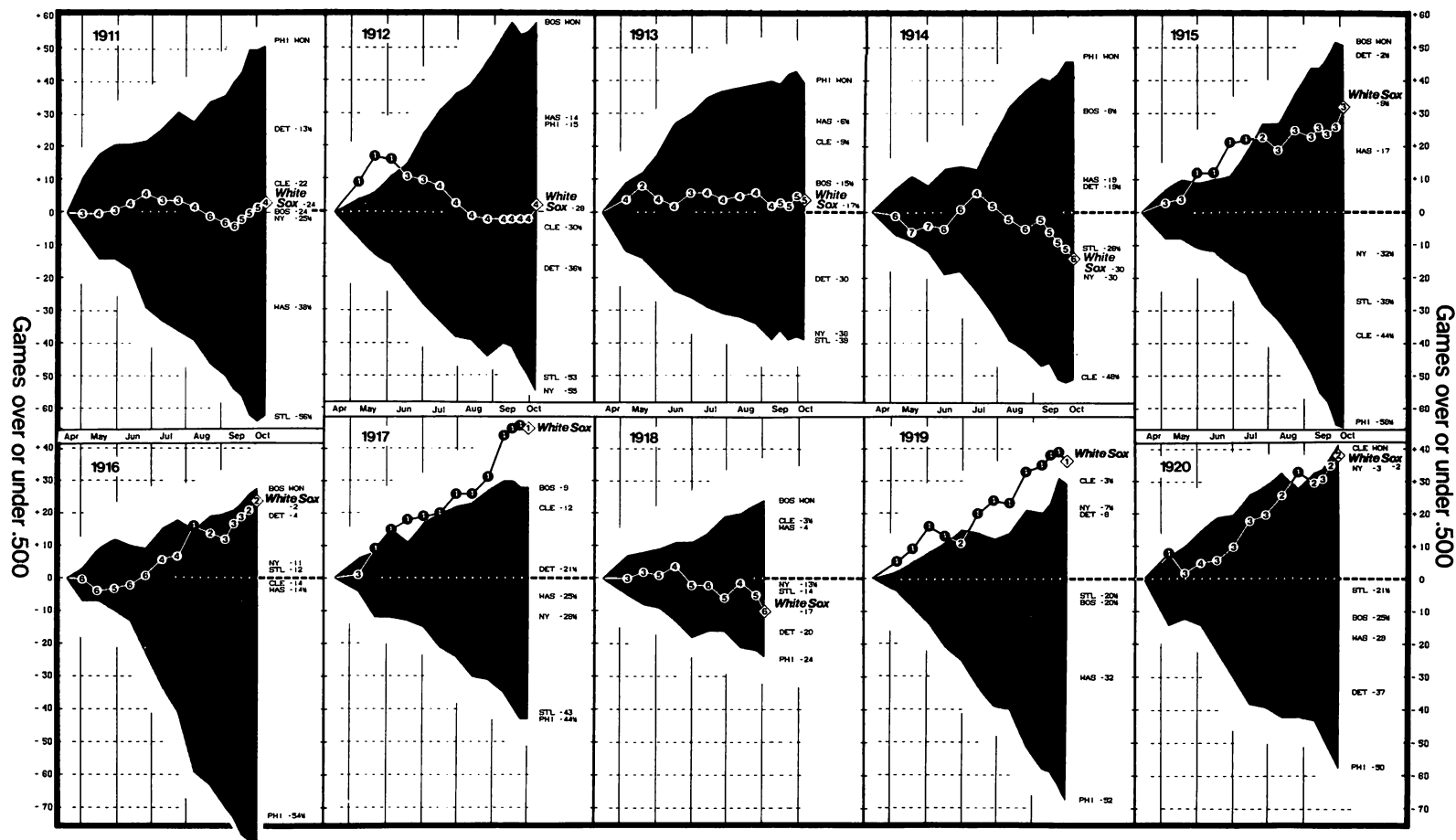


FIGURE 12-7. A rank-and-range multigraph. (Reprinted with permission of Macmillan Publishing Company from *SOX: The Complete Record of Chicago White Sox Baseball* by Richard Lindberg, graphics by John W. Davenport. Copyright © 1984 by Macmillan Publishing Company, a division of Macmillan, Inc.)

1979 SEASON SERIES SUMMARIES



FIGURE 12-8. The round-robin taken seriously—a multiple-pie graph. (Reprinted with permission of First Impressions Publishing Company, Madison, Wisconsin, from *Baseball Graphics '79 Sampler*, copyright © 1979 by John Warner Davenport.)

was done by hand, and the hardest part was the pies themselves. But a program like PIECHART, modified to paint one section of each pie black, could handle all of that tedium, including some tricky calculations in dividing up the pies. To work best, this scheme should include drawing the pies around five inches in diameter, reducing them photographically to get rid of rough edges, and using dry-transfer white and black numbers to finish them. In a multiple-pie chart, the fact that you can't do labeling with PIECHART doesn't hurt a bit.

Montages

Perhaps the ultimate form of multigraph in the graphics world is the montage. Some might even say that the montage

is the ultimate form of “cheating,” in both the graphics and computer-graphics worlds. You have to admit that montages involve cutting corners, but nothing beats them as a way of “putting it all together.” And figure 12-9 suggests that if a picture is worth a thousand words, maybe a montage is worth a whole book.

Printer Graphics in the Near Future

Progress in printer graphics or even printer-aided graphics in microcomputing will hardly be measured by how big we can make a multigraph or how cleverly we can assemble a montage. And yet the examples in this chapter are evidence of some progress, in the sense that just two or three years ago it was pretty hard to find any printout from a dot-matrix printer that was worth the effort of manual finishing touches.

But there seems to be a continual insinuation, in current commentary in the microcomputing world, that printer graphics will show very little further progress because resolution in screen graphics is fast catching up. As soon as screen displays go a bit beyond the Macintosh’s 512 by 342 pixels or the Tandy 2000’s 640 by 400 in their resolution, it is implied, dumping their contents to a printer of today’s caliber will come as close to professional-quality printing as any direct programming of a printer, and will be as good as we would want anyway. Furthermore, pictorial graphics will progress in leaps and bounds from the advent of new screen-oriented peripheral devices such as optical scanners for automatic digitizing.

It’s easy to see why nearly all the development in microcomputer graphics goes into screen-oriented devices and software. Even printer graphics buffs may not need hard copy for more than ten per cent of the graphics they produce, and why waste so much paper and time making printouts that’ll end up in the wastebasket? But there seems to be an overconcentration on screen-display graphics these days and continued neglect of the capabilities of printers. Why not use both to their fullest extent?

For that matter, *will* the computers catch up to the printers? It’s doubtful, if you look at the advances in printer design coming forth every month nowadays. Since 1981, in dot-addressable printers, we’ve come from a rough average



FIGURE 12-9. If all else fails, you can just toss all your printouts into a pile “at random”—you might even win an artistic award for composition of a montage.

resolution of 72 (vertically) by 110 (horizontally) dots per inch (e.g., the Epson MX-80 and Radio Shack LPVIII printers) to 18-pin printers (such as the Radio Shack DMP-430) having 144 dots per inch vertically in “TRS-80 mode” and up to 240 dots per inch horizontally in “IBM mode.” (The fact that the DMP-430 is able to shift from TRS-80 to IBM-Epson control and dot codes with the flip of a DIP switch or a software command is itself a significant advance that we can only hope will be imitated.) And the 24-pin printers (there are now several of them) increase the resolution to as much as 216 by 240 (e.g., the Epson LQ-1500 in quadruple-density bit-image mode).

If you're impressed by the new laser printers such as Apple's LaserWriter, which can produce typography that is indistinguishable by the naked eye from that produced by much more expensive typesetting machines (see the dazzling examples in *Macworld*, February 1985), consider the fact that the LaserWriter's resolution is not that much further ahead—300 by 300. When that is translated to 90,000 dots in each square inch, it seems mind-boggling, but so is the LQ-1500's 51,840. So we already have printers costing under fifteen hundred dollars that are approaching seven-thousand-dollar laser printers in fineness of detail.

With these newer impact printers, there's a price you pay for the high resolution—instead of 256 dot patterns as on an eight-pin printer you now have 65,536 different patterns on an eighteen-pin printer and 16,777,216 on a twenty-four-pin printer. When the programming problems presented by this are licked, these printers should make the graphics in a book like this one look primitive and screen dumps even more so. Better labeling of graphs, better-looking connecting lines in line graphs, pictures that would make AP Laserphotos seem obsolete, smaller dot patterns for tints and textures, better typography (though still not up to the LaserWriter's), more detailed diagrams and maps—all these things should become routine, and make it possible for the printer to do a much larger proportion of the work in computer-aided graphics, well before the end of the 1980s. Another advance, already visible in comparing the DMP-430 with the DMP-400, will be better consistency of line feeds in their exact size. Colored-ribbon and ink-jet printers will probably have less garish hues and more ways of mixing colors.

We can also anticipate automatic digitizers for directly programming printers. That is, instead of a Thunderscan creating digital codes for a MacPaint drawing on the Macintosh's screen that is dumped to an Imagewriter, we'll have an image scanner that sends many more codes to the computer and, via a printer graphics program that bypasses the screen, helps us produce truly photographic quality in pictorial material. And once the codes are in digital form, of course, they can be sent from computer to computer (and printer), across the hall or across the country. Scanning digitizers will not be the only new peripheral for direct-programmed printer graphics; devices working on quite different principles can be expected for making graphs, typography, maps, and so on easier to produce at a printer's highest resolution. And as evidenced throughout this book, one needn't have formal training in computer science or graphic design to get the most productivity—and the most enjoyment—out of a personal-computer system for hard-copy graphics.

After all these likely innovations, we may note wistfully, there may be nothing left of the pioneering spirit of the early bit-image enthusiasts—people such as Keller, Kater, and Kalinowski. And to us dedicated do-it-yourself digitizers, these things threaten to spoil all the fun!

APPENDIX A

Data Codes for Two LINEDRAW-Type Programs

PROGRAM A-1. DENNIS. Cartoon drawing.

```
10 'PROGRAM A-1: "DENNIS" -- CARTOON DRAWING (TRS-80 PRINTERS)
20 CLEAR 1000
30 LPRINT CHR$(30);CHR$(27);CHR$(20); 'SET CONDENSED PRINT DENSITY
40 LPRINT CHR$(27);CHR$(31); 'BOLD PRINTING
50 LPRINT CHR$(18); 'ENTER GRAPHIC MODE
100 FOR R=1 TO 23 'LOOP FOR 23 ROWS
110 READ N: IF N=999 THEN GOTO 160 'READ CODE #, LINE FEED IF 999
120 IF N=333 THEN READ LS: GOSUB 480: GOTO 110
130 IF N>=0 AND N<128 THEN LPRINT CHR$(128+N);: C=C+1: GOTO 110 'PRINT CODE
    # IF NO NEGATIVE NUMBER
140 READ M 'CODE # WAS NOT POSITIVE OR ZERO, SO READ NEXT
150 FOR Z=1 TO -N: LPRINT CHR$(128+M);: C=C+1: NEXT Z: GOTO 110
160 LPRINT CHR$(13);: C=0: NEXT R
170 END
180 DATA -226,112,999
190 DATA -3,127,-220,0,-3,127,999
200 DATA -3,127,-220,0,-3,127,999
210 DATA -3,127,-220,0,-3,127,999
220 DATA -3,127,-53,0,64,64,96,96,-2,112,120,56,56,-4,28,-5,16,-4,8,-5,12,-12,6,
    -6,7,-3,6,-5,4,-5,8,-6,16,32,32,-3,64,-98,0,-3,127,999
230 DATA -3,127,-41,0,64,96,96,-2,112,56,60,28,14,14,7,7,-3,3,1,1,-15,0,-3,64,-4
    ,32,-4,16,-7,8,-9,4,-11,66,-11,68,-3,69,70,70,-3,74,-3,76,-2,108,88,24,16,80
    ,16,-80,0,-3,127,999
240 DATA -3,127,-18,0,-3,112,96,96,-3,64,-4,0,64,64,96,96,48,56,28,12,14,7,3,3,1
    ,1,-3,0,-3,64,32,32,-3,16,-4,8,-4,4,-4,2,1,-3,65,33,97,32,32,-3,16,-5,8,-5,4
    ,-4,2,-9,1,-3,0,32,56,28,14,3,1,0,-3,64,-4,32,-3,16
```

250 DATA -3,8,4,4,66,66,33,17,16,8,8,4,4,2,3,1,1,2,-80,0,-3,127,999
 260 DATA -3,127,-17,0,112,15,-7,0,1,1,2,3,3,7,3,1,-6,0,8,4,4,66,66,33,33,32,-3,1
 6,-3,8,-4,4,-5,2,-5,1,-4,0,1,3,3,70,70,76,68,64,-7,32,-4,16,-4,8,-4,4,-6,2,-
 5,1,65,64,79,48,32,-3,80,16,8,8,4,4,2,2,1,-12,0,-3,64,-79,0,-3,127,999
 270 DATA -3,127,-17,0,127,-17,0,2,10,6,18,18,-3,35,36,-7,68,66,-6,2,-9,66,-8,65,
 64,64,-9,32,-5,16,-5,8,-4,4,68,68,-4,34,33,65,97,97,96,112,112,48,8,0,0,1,1,
 2,2,4,12,8,24,16,48,96,96,64,0,64,32,16,8,6,1,1,-3,0,1,6,24,96,-75,0,-3,127,
 999
 280 DATA -3,127,-16,0,124,3,0,0,64,96,96,-4,48,112,-7,120,56,-8,0,96,96,-6,0,-6,
 1,-9,0,96,0,112,15,-9,0,-3,32,16,16,112,-3,120,100,4,-4,0,56,68,2,1,1,-3,0,6
 4,0,64,-3,0,1,71,59,-8,0,16,0,0,4,-3,0,64,0,0,1,1,3,6,12,24,-2,112,64,-5,0,1
 ,126,-74,0,-3,127,999
 290 DATA -3,127,-16,0,15,112,48,6,3,1,-6,0,1,3,3,67,51,15,1,-7,0,32,48,32,-12,0,
 64,-4,32,-3,0,112,14,1,112,15,-15,0,-3,3,1,-3,0,64,-3,0,1,6,4,-3,8,7,4,7,7,4
 ,-3,2,1,-26,0,7,127,126,32,32,16,12,3,-7,0
 300 DATA -3,64,96,96,32,32,96,-11,32,-3,64,32,-3,16,-4,8,16,96,-4,0,64,64,-3,32,
 -8,16,96,-18,0,-3,127,999
 310 DATA -3,127,-17,0,3,-11,0,96,28,3,-5,0,120,124,-12,0,64,32,8,4,2,1,64,64,32,
 16,16,8,74,33,24,4,3,-17,0,32,-5,0,32,-17,0,96,-4,64,96,32,32,16,8,6,-16,0,6
 4,120,127,3,0,0,64,96,48,24,12,6,2,3,-3,1,-6,0
 320 DATA 1,6,120,-10,0,14,17,-3,16,8,24,52,64,0,0,8,7,4,2,1,1,32,-7,80,72,72,72,
 68,68,65,64,-16,0,-3,127,999
 330 DATA -3,127,-30,0,15,-6,0,31,127,124,-2,120,112,96,-6,64,0,65,64,67,35,97,97
 ,48,16,8,4,2,1,-25,0,8,-7,0,4,-11,0,56,7,-3,0,-3,1,-17,0,64,112,56,30,15,7,0
 ,112,14,7,3,-18,0,64,71,120,-5,64,-11,0
 340 DATA 3,15,112,-7,0,32,96,96,-6,32,64,64,32,32,33,30,-14,0,-3,127,999
 350 DATA -3,127,-40,0,1,1,67,35,19,11,7,7,3,3,1,-73,0,64,64,96,06,48,24,96,70,35
 ,17,16,-4,8,7,4,68,36,28,-15,0,28,35,65,64,-3,0,-3,64,95,0,0,64,0,2,-3,0,16,
 -3,0,127,28,-5,0,32,64,64,0,1,1,-3,2,-3,4,8,112,-17,0,-3,127,999
 360 DATA -3,127,-40,0,28,35,64,-36,0,64,96,48,56,-3,48,-8,96,-8,48,-3,24,-4,88,-
 4,120,-4,124,-3,126,-8,127,15,6,1,-3,0,32,16,8,4,4,2,65,-3,33,17,33,33,65,66
 ,68,72,48,-11,0,1,1,15,112,-14,0,112,127,15,-8,0
 370 DATA 1,1,127,1,-6,2,1,-17,0,-3,127,999
 380 DATA -3,127,-43,0,1,3,2,6,4,4,12,-4,8,-16,16,72,40,40,20,20,10,10,11,9,17,-7
 ,16,-4,32,64,96,96,112,-2,120,124,-2,126,-14,127,-4,95,79,71,39,39,35,67,65,
 49,15,-9,0,2,1,1,0,64,-3,32,-6,16,17,60,64,-14,0,15,112,-7,0
 390 DATA 64,96,112,120,6,3,-12,0,127,-25,0,-3,127,999
 400 DATA -3,127,-67,0,126,1,1,1,2,58,66,2,2,2,-4,4,68,40,24,24,-3,8,-3,4,2,2,3
 ,2,2,-6,1,-34,0,65,97,80,104,4,4,2,7,122,4,4,24,32,64,64,58,114,2,-3,4,-8,8,
 24,104,-3,4,6,9,4,126,-5,127,96,64,-12,0,3,124,-24,0,-3,127,999
 410 DATA -3,127,-67,0,3,4,8,48,64,0,0,1,2,4,8,112,8,6,1,-13,0,64,32,16,8,72,32,9
 6,0,16,96,-21,0,64,32,112,-6,8,28,98,1,0,0,31,99,124,-4,0,63,64,-6,0,3,12,11
 2,-11,0,3,28,96,0,3,-7,127,79,64,-3,32,16,16,8,8,-3,4,2,2,1,-24,0,-3,127,999
 420 DATA -3,127,-72,0,1,-6,2,15,48,64,-14,0,15,46,43,36,18,9,4,3,-3,0,-4,64,32,9
 6,96,112,16,16,-3,8,4,4,2,2,1,1,0,0,127,-7,0,15,112,-3,0,63,64,63,64,-3,0,31
 ,96,-8,0,7,120,-12,0,7,24,103,-6,127,96,-38,0,-3,127,999
 430 DATA -3,127,-82,0,1,6,28,-11,120,124,108,12,12,6,6,62,-2,126,-4,127,120,0,0,
 -3,127,-12,0,124,3,-8,0,127,-4,0,31,96,31,96,-3,0,7,120,-9,0,3,124,-13,0,3,1
 3,113,-4,3,-38,0,-3,127,999
 440 DATA -3,127,-85,0,1,7,-11,127,126,120,96,0,7,31,-5,127,31,31,56,48,17,127,15
 ,-10,0,96,31,-9,0,63,64,-4,0,7,120,7,120,-4,0,7,56,64,0,24,96,0,0,24,96,1,14
 ,112,-14,0,7,120,-40,0,-3,127,999
 450 DATA -3,127,-220,0,-3,127,999
 460 DATA -226,7,999

PROGRAM A-2. USAMAP. Map of the United States.

```

10 'PROGRAM A-2: "USAMAP" -- MAP OF THE LOWER 48 UNITED STATES (TRS-80 DMP-400)
20 CLEAR 5000
30 INPUT "BLACK-ON-WHITE (B) OR WHITE-ON-BLACK (W)";BW$
40 LPRINT CHR$(30);CHR$(27);CHR$(20)
50 LPRINT CHR$(18)
60 FOR R=120 TO 660 STEP 10
70 READ N: IF N=999 THEN GOTO 140
80 IF BW$="B" AND N>=0 THEN LPRINT CHR$(128+N);: GOTO 70
90 IF N>=0 THEN LPRINT CHR$(255-N);: GOTO 70
100 READ M
110 IF BW$="B" THEN LPRINT STRING$(-N,128+M);: GOTO 70
120 LPRINT STRING$(-N,255-M);
130 GOTO 70
140 LPRINT CHR$(13);: NEXT R
150 FOR N=1 TO 5: LPRINT CHR$(13);: NEXT
160 LPRINT CHR$(30);CHR$(27);CHR$(18);"    DRAWN BY RADIO SHACK DMP-400 DOT MATR
IX PRINTER AND TRS-80 MODEL III MICROCOMPUTER"
170 LPRINT CHR$(13);: LPRINT "    (Courtesy of First Impressions Pub. Co., P.
O. Box 9073, Madison, WI, 53715)";CHR$(27);CHR$(19)
180 END
190 DATA -2,127,-200,0,-200,0,-200,0,-200,0,-96,0,-2,127,999
200 DATA -2,127,-91,0,3,-2,98,34,36,68,-2,8,16,32,-2,64,-6,0,64,17,10,12,-2,0,1,
-3,2,1,62,68,-6,4,-5,8,-5,16,-5,32,-5,64,-250,0,-250,0,-248,0,-2,127,999
210 DATA -2,127,-90,0,124,3,-4,0,1,2,2,4,4,8,8,17,17,-2,34,33,65,-2,64,-5,0,96,3
2,28,4,7,-26,0,-5,1,-5,2,-5,4,-5,8,-5,16,-5,32,-6,64,-250,0,-250,0,-213,0,-2
,127,999
220 DATA -2,127,-90,0,127,-20,0,65,34,19,33,15,64,53,71,-65,0,-7,1,2,2,114,14,-3
,2,-8,4,8,64,56,-5,8,-8,16,-8,32,-8,64,-250,0,-250,0,-96,0,0,64,32,24,8,16,9
6,-2,64,-2,32,-2,16,8,16,-3,8,-6,16,32,64,-36,0,-2,127,999
230 DATA -2,127,-90,0,15,48,32,64,64,-12,0,24,34,74,53,37,98,68,34,78,48,24,7,-7
1,0,64,56,7,-13,0,120,7,-30,0,-9,1,-9,2,-10,4,-10,8,-10,16,-11,32,-11,64,-25
0,0,-250,0,-23,0,64,32,16,8,7,-24,0,1,6,56,64,-32,0,-2,127,999
240 DATA -2,127,-88,0,96,-2,16,18,-2,22,14,1,-14,0,-5,1,-73,0,96,56,3,-15,0,23,9
6,-100,0,-12,1,-13,2,-14,4,-16,8,-3,16,112,-16,16,-25,32,-30,64,-48,0,124,4,
4,24,96,-84,0,96,48,8,112,2,93,81,50,26,10,-8,2,-8,4
250 DATA 12,8,4,8,16,32,64,-223,0,15,112,-30,0,1,6,24,96,-29,0,-2,127,999
260 DATA -2,127,-87,0,112,19,18,10,18,36,36,40,40,-4,32,64,64,-84,0,112,14,1,-18
,0,127,-157,0,64,63,-71,0,-22,1,127,-25,1,-3,0,1,2,4,12,-4,16,-6,8,16,32,-4,
64,-6,32,64,64,-49,0,64,32,24,4,14,10,4,3,-3,0,2,1,1
270 DATA -26,0,1,6,8,16,16,-9,32,-4,16,8,120,8,-150,0,-53,0,62,65,-34,0,3,-3,4
,8,16,32,32,-3,64,-18,0,-2,127,999
280 DATA -2,127,-84,0,64,48,12,3,-14,0,1,97,18,12,-78,0,120,7,-21,0,1,2,4,8,8,16
,32,32,64,-149,0,127,-95,0,63,64,-56,0,1,2,2,4,8,8,16,16,8,8,-4,4,8,-4,16,8,
8,4,2,2,-3,4,-5,8,16,-6,8,72,72,48,32,16,8,8,6,3,-60,0,1,2,4,8,8,16,16,96,64
,-194,0
290 DATA 96,30,1,-44,0,1,2,5,5,9,41,81,1,2,2,1,-8,0,-2,127,999
300 DATA -2,127,-80,0,64,56,4,2,1,-17,0,7,-8,8,4,2,-5,4,8,8,16,16,-5,32,64,64,-4
,0,-5,64,-4,32,-3,16,-3,8,-2,16,-5,32,-6,64,-22,0,126,1,-31,0,3,4,4,8,112,-1
43,0,112,15,-96,0,1,126,-81,0,64,32,16,8,8,-4,4,-3,2,1,1
310 DATA -19,0,64,64,32,16,8,8,4,4,68,36,20,12,-44,0,28,35,32,64,-184,0,64,48,-4
,16,102,6,1,-40,0,96,32,32,64,32,16,8,8,4,4,2,1,-13,0,-2,127,999
320 DATA -2,127,-77,0,112,12,3,-49,0,-4,1,-28,0,-7,1,-5,2,-5,4,-5,8,63,64,-33,0,

```

125,2,1,-142,0,96,31,-97,0,112,15,-73,0,64,32,16,16,8,4,2,1,0,64,-17,0,-3,64
 , -3,32,-4,16,8,4,2,1,0,0,16,24,20,18,17,17,16,16,8,8,16,32,32,64,-17,0
 330 DATA 64,32,-13,16,8,-3,4,60,64,64,0,0,64,95,80,16,16,8,-3,16,96,-18,0,-3,64,
 -6,0,64,-124,0,-7,64,-7,32,-7,16,8,8,126,1,-6,0,7,24,112,-27,0,126,1,2,12,16
 ,16,28,4,4,10,22,9,-26,0,-2,127,999
 340 DATA -2,127,-73,0,64,48,8,6,1,-106,0,64,33,18,12,-28,0,56,70,1,0,64,64,32,32
 ,96,-138,0,127,-98,0,3,4,8,48,64,-61,0,112,16,8,8,4,10,18,17,-4,8,4,4,2,1,1,
 0,1,30,-5,16,16,8,56,72,8,-3,4,2,1,-31,0,1,2,-8,4,-4,8,4,2,1,-15,0
 350 DATA 64,64,32,32,-3,64,1,1,0,0,96,33,33,45,83,81,86,40,40,8,19,13,17,41,41,4
 2,72,10,82,18,-5,82,81,80,80,96,81,81,74,73,69,85,68,5,5,3,15,10,10,42,116,-
 5,4,2,1,1,1,2,12,16,96,-75,0,64,32,96,32,16,16,8,-6,4,-6,2,-7,1,3,60,64,
 360 DATA -20,0,1,126,-9,0,1,14,56,64,-15,0,64,0,112,124,16,16,12,2,3,-38,0,-2,12
 7,999
 370 DATA -2,127,-66,0,112,8,8,4,2,1,1,-102,0,64,32,16,8,4,2,1,1,-35,0,3,1,-3,0,1
 ,6,8,16,96,-134,0,120,67,-3,64,-100,0,127,-59,0,64,127,-27,0,1,-20,2,-6,4,8,
 8,48,64,-15,0,64,32,24,4,8,8,16,8,36,44,20,8,8,-6,4,2,2,1,1,1,-7,0,1,33,82,1
 0,10,11,11,10
 380 DATA -3,8,16,16,32,32,64,64,-5,0,1,0,1,-16,0,1,1,2,4,2,1,1,0,48,80,16,32,32,
 64,-6,0,1,1,-4,2,4,60,32,64,64,-57,0,64,32,32,16,72,36,26,5,2,1,-26,0,7,120,
 -19,0,64,63,-12,0,1,6,24,96,-8,0,124,2,1,2,3,1,3,-44,0,-2,127,999
 390 DATA -2,127,-59,0,64,32,16,16,8,8,4,3,-106,0,28,99,-52,0,15,16,32,64,-29,0,-
 9,64,-92,0,96,31,-4,0,-31,1,-34,2,-33,4,36,84,15,-52,0,24,36,36,66,2,1,1,-59
 ,0,1,0,1,31,32,32,64,-4,0,64,48,8,6,65,32,24,100,30,-17,0,64,96,-2,0,112
 400 DATA 12,-3,4,2,2,1,-13,0,-5,1,2,4,100,24,-25,0,1,3,4,88,33,6,8,4,8,112,64,32
 ,16,16,8,4,48,-2,64,66,74,114,2,3,2,-40,0,64,96,96,-3,16,24,4,4,-3,2,18,25,2
 1,21,20,34,34,33,64,-32,0,7,8,8,16,96,-12,0,120,6,1,-18,0,1,-3,2,68,36,124,3
 ,-50,0,-2,127,999
 410 DATA -2,127,-59,0,111,16,-111,0,96,24,7,-57,0,7,8,-7,16,-3,8,-8,16,8,4,4,2,4
 ,8,16,96,62,1,-8,0,-15,1,-15,2,-15,4,-15,8,-15,16,-16,32,127,-105,0,1,2,4,12
 0,-51,0,120,5,2,-66,0,65,34,20,76,34,17,8,4,98,17,12,3,-14,0,96,16,8,4,2,1,0
 ,15,16,12,3
 420 DATA -28,0,1,126,-26,0,64,57,6,-39,0,-3,64,-3,32,-4,16,8,8,4,4,2,2,4,8,16,16
 ,-3,8,-3,9,5,6,2,-7,0,64,48,14,1,1,-39,0,7,24,96,-9,0,7,120,-3,0,-5,64,-4,32
 ,-4,16,-4,8,4,3,0,64,33,18,12,-49,0,-2,127,999
 430 DATA -2,127,-58,0,126,1,-110,0,112,15,-86,0,64,63,-100,0,112,15,-108,0,127,-
 51,0,7,8,16,32,64,64,-62,0,4,5,2,1,0,0,112,12,3,-18,0,127,-33,0,64,32,16,8,4
 ,2,1,64,32,16,16,32,32,48,8,-3,4,8,16,96,-11,0,3,4,56,64,-28,0,96,16,8,8,4,4
 ,2,2,-3,1
 440 DATA -7,0,-6,64,-6,32,64,32,96,32,32,16,16,-4,8,4,4,2,2,1,-47,0,3,124,-4,4,2
 ,2,2,2,-4,1,-20,0,3,-5,2,4,8,16,96,-3,0,8,8,120,112,-37,0,-2,127,999
 450 DATA -2,127,-58,0,95,34,-3,2,-5,4,-6,8,-7,16,-7,32,-7,64,-73,0,112,15,-87,0,
 96,31,-100,0,112,15,-109,0,127,-58,0,1,1,2,2,4,4,4,8,8,8,16,16,32,32,64,64,-
 51,0,1,126,-18,0,96,31,-33,0,3,-3,4,2,1,-14,0,31,16,32,96,-8,0,64,48,12,3,-2
 7,0
 460 DATA 12,11,24,16,16,72,-3,8,4,68,56,24,36,68,-3,2,1,1,-11,0,2,5,4,3,-61,0,12
 7,64,-5,32,-5,16,-5,8,-4,4,12,20,100,4,4,2,2,4,68,8,16,64,0,64,32,16,12,4,28
 ,32,35,-5,20,23,72,79,64,64,-34,0,-2,127,999
 470 DATA -2,127,-56,0,96,24,7,-36,0,-7,1,-6,2,-6,4,-6,8,80,48,-3,16,-6,32,-7,64,
 -27,0,64,56,7,-88,0,120,7,-101,0,127,-111,0,127,-72,0,7,24,96,-49,0,124,6,1,
 -18,0,3,12,16,32,64,-54,0,65,33,28,0,124,2,2,1,1,-9,0,-3,64,-6,32,-3,64,48,4
 0,72
 480 DATA -6,4,2,2,-3,1,0,1,1,65,33,32,16,8,4,3,-38,0,-6,64,-6,32,-3,16,-3,32,64,
 -23,0,7,56,64,-18,0,64,64,35,44,-3,16,8,15,2,2,3,1,-8,0,2,5,5,6,0,0,-3,1,-35
 ,0,-2,127,999

490 DATA -2,127,-52,0,96,24,4,2,1,-62,0,112,14,1,-16,0,-7,1,-7,2,-7,4,-5,8,15,8,
8,-7,16,-7,32,-7,64,-66,0,120,7,-102,0,126,1,-6,64,-102,0,56,44,1,-12,8,-25,
4,-25,2,-13,1,3,124,-48,0,1,126,-23,0,64,63,-47,0,96,16,80,20,18,18,17,8,4,3
, -4,0

500 DATA -3,64,32,48,8,4,2,2,1,-12,0,-4,1,0,64,64,32,16,16,8,8,4,4,2,29,97,-5,64
, -6,32,-7,16,-6,8,-7,4,-6,2,-7,1,-19,0,1,1,6,8,8,-4,16,80,48,32,32,-9,64,-3,
0,48,72,79,32,-4,16,72,72,4,68,36,36,34,34,18,18,9,73,89,88,48,40,24,8,-59,0
, -2,127,999

510 DATA -2,127,-50,0,24,38,65,-64,0,96,30,1,-69,0,-7,1,-7,2,-4,4,124,4,4,-8,8,-
7,16,-7,32,-9,64,-11,0,64,62,1,-103,0,127,-7,0,-25,1,-24,2,-23,4,-5,8,16,16,
-5,32,64,32,32,-7,16,32,-3,64,0,-4,64,60,3,-77,0,1,2,2,4,8,8,16,16,32,64,64,
-20,64,-19,32,127

520 DATA -21,0,120,7,-45,0,96,30,1,31,16,16,-5,8,16,28,2,1,1,-14,0,64,32,16,16,8
, -3,4,2,2,1,15,113,-82,0,112,14,1,-12,0,96,16,83,60,24,62,69,68,34,34,33,-2,
17,16,17,9,4,4,2,2,1,1,-67,0,-2,127,999

530 DATA -2,127,-53,0,1,2,108,16,-58,0,96,28,3,-88,0,126,1,-33,0,-8,1,-3,2,98,31
, -104,0,127,-105,0,1,-4,0,-3,1,6,24,32,64,-82,0,1,1,4,8,112,-36,0,3,4,24,32,
64,-11,0,64,48,8,70,65,-10,64,-12,32,-6,16,112,-5,16,-12,8,12,19,32,32,64,64
, -3,32,-3,16

540 DATA 32,64,32,16,16,8,8,-4,4,-3,2,-5,1,-12,0,1,14,112,-81,0,1,6,4,8,16,32,64
, -6,0,1,1,3,-4,2,4,8,112,-80,0,-2,127,999

550 DATA -2,127,-53,0,64,62,1,-57,0,96,28,3,-88,0,64,63,-45,0,112,15,-105,0,127,
-116,0,3,124,-83,0,96,16,8,15,-40,0,1,14,116,4,4,-3,2,-6,1,-30,0,3,124,-62,0
, 1,30,96,-79,0,96,16,8,8,4,2,1,-12,0,96,24,7,-80,0,-2,127,999

560 DATA -2,127,-52,0,16,45,66,-56,0,96,28,3,-89,0,124,3,-46,0,7,4,4,4,-14,8,-11
, 16,96,16,16,-14,32,-14,64,-47,0,127,-118,0,7,8,112,-71,0,48,40,72,4,-4,2,1,
-46,0,7,120,-41,0,1,126,-62,0,96,31,124,-52,0,-7,64,-7,32,-6,16,-4,8,120,8,3
1,104,16,32

570 DATA -11,0,96,16,8,7,-83,0,-2,127,999

580 DATA -2,127,-55,0,1,2,30,96,-50,0,64,60,3,-89,0,64,62,1,-75,0,112,15,-30,0,-
15,1,-15,2,-16,4,8,15,-14,8,-15,16,112,-91,0,127,-70,0,64,48,12,3,-53,0,7,12
0,-41,0,63,64,-60,0,113,14,0,7,56,64,0,-7,64,-7,32,16,112,-5,16

590 DATA -4,8,72,-5,36,20,20,-3,10,-3,18,-3,33,65,-5,1,-10,0,64,32,16,16,8,12,48
, 64,-6,0,7,120,0,0,1,6,8,25,97,1,-3,2,4,8,8,7,-87,0,-2,127,999

600 DATA -2,127,-58,0,31,32,64,-5,0,96,16,96,-41,0,1,2,4,8,16,96,-84,0,112,15,-7
6,0,120,7,-107,0,96,31,-91,0,1,2,4,4,8,120,-18,8,-25,4,-15,2,4,8,8,16,16,124
, 3,-58,0,3,124,-42,0,31,96,-47,0,-4,64,32,32,24,8,4,4,2,1,-5,0,1,1,-16,0

610 DATA 7,4,2,2,-5,1,-4,0,64,56,6,2,2,-6,4,8,12,11,-4,8,4,8,8,-3,4,120,-5,0,1,9
7,17,39,76,48,99,4,8,120,-5,0,31,96,-4,64,32,33,33,34,34,20,20,124,-87,0,-2,
127,999

620 DATA -2,127,-62,0,97,17,10,26,97,0,0,127,-47,0,1,2,4,8,16,32,64,64,-74,0,124
, 3,-76,0,120,7,-108,0,96,31,-55,16,-41,32,33,34,36,40,48,-6,64,-53,0,1,14,16
, 96,-57,0,3,124,-42,0,15,112,0,-3,64,-37,0,96,-3,16,18,13,-40,0,120,-3,4,-3,
2,1,-22,0

630 DATA 15,16,16,48,40,40,72,8,8,17,18,20,21,43,58,80,103,8,14,12,16,14,2,5,4,1
20,-7,0,96,31,-88,0,-2,127,999

640 DATA -2,127,-62,0,15,48,32,32,64,1,2,3,-54,0,1,2,4,8,16,32,64,-66,0,64,63,-7
7,0,124,3,-109,0,127,-107,0,3,28,32,64,64,-53,0,1,1,2,2,4,4,8,8,16,32,64,-47
, 0,1,126,-35,0,64,32,-3,16,12,2,2,1,-3,0,3,-6,4,8,-4,16,-4,8,16,16,32,64,32,
32

650 DATA 16,8,4,8,8,16,-3,32,96,16,16,8,12,3,-34,0,64,64,32,16,24,7,-6,1,2,3,-36
, 0,1,-3,2,-3,1,113,73,106,78,64,-6,0,99,28,0,64,48,12,3,-90,0,-2,127,999

660 DATA -2,127,-65,0,112,16,17,28,-62,0,1,2,4,8,16,32,64,-59,0,112,15,-76,0,124
, 3,-109,0,112,15,-112,0,1,1,126,-61,0,1,2,-6,4,120,-37,0,64,48,12,3,-28,0,10

0,28,4,4,2,1,-43,0,1,6,8,112,-33,0,126,1,-57,0,105,24,57,2,2,1,-6,0,7,12,3,-94,0,-2,127,999

670 DATA -2,127,-65,0,1,2,28,96,-69,0,1,2,4,8,16,32,64,-51,0,124,3,-76,0,124,3,-110,0,127,-115,0,127,-66,0,12,19,32,64,64,-33,0,12,19,32,64,0,64,64,32,64,0,0,64,-4,32,64,64,32,96,64,96,16,8,4,2,4,-3,8,4,2,2,1,-54,0,1,2,4,8,16,16,80,-3,32,96

680 DATA -11,0,64,32,32,-3,16,8,4,4,2,1,-60,0,1,14,-3,16,72,72,68,68,66,66,65,33,33,126,28,96,-91,0,-2,127,999

690 DATA -2,127,-68,0,1,30,96,-74,0,1,2,4,8,16,32,64,-42,0,64,63,-77,0,124,3,-110,0,96,31,-116,0,127,-70,0,1,1,2,4,4,8,8,16,16,96,-18,0,64,-3,32,120,4,2,1,1,-5,0,1,-74,0,64,32,16,14,-3,2,1,1,-6,0,1,1,2,2,-4,1,2,2,1,1,-12,0

700 DATA -9,64,-9,32,-9,16,-9,8,-9,4,-9,2,-4,1,57,81,33,65,64,48,64,32,32,16,8,16,8,9,10,13,9,20,8,16,96,64,-86,0,-2,127,999

710 DATA -2,127,-70,0,1,2,4,8,16,32,64,-74,0,1,2,4,8,16,32,64,-35,0,124,3,-12,2,-13,4,-13,8,-13,16,-13,32,-13,64,112,15,-111,0,127,-117,0,127,-78,0,6,9,8,16,32,16,16,8,8,112,16,8,8,16,16,32,16,8,30,33,-62,0,-10,64,-5,32,-3,48,40,24

720 DATA -4,20,-3,18,17,-10,8,-7,4,68,60,4,4,-9,2,-9,1,-58,0,3,4,8,-4,4,-3,2,-3,1,6,56,49,103,17,18,12,0,3,1,0,4,8,24,96,-79,0,-2,127,999

730 DATA -2,127,-76,0,7,8,16,32,64,-78,0,1,6,8,16,32,64,-15,0,-4,64,-7,0,96,28,3,-77,0,112,15,-15,1,-16,2,-16,4,-16,8,-34,16,112,-14,16,31,-50,32,-67,64,127,-85,0,64,32,31,16,-16,0,96,16,-13,8,-13,4,-13,2,-14,1,-36,0,64,32,32,-3,16,-3,8,6,1,-82,0

740 DATA 4,12,-3,10,34,98,34,-3,2,66,2,2,3,0,0,64,32,0,8,8,4,3,1,-78,0,-2,127,999

750 DATA -2,127,-78,0,112,12,3,-84,0,1,2,12,16,32,64,-8,0,127,-4,0,1,1,2,2,4,2,1,-79,0,112,15,-97,0,120,71,-47,64,-85,0,63,80,-36,16,-37,8,96,24,-5,0,96,24,4,5,5,-10,2,-10,1,-78,0,64,64,112,8,4,4,-6,2,3,-91,0,12,74,-3,42,73,73,68,70,36

760 DATA 98,83,44,19,8,4,0,1,1,-85,0,-2,127,999

770 DATA -2,127,-79,0,-7,1,-5,2,-5,4,8,8,16,32,64,-70,0,1,2,4,8,16,32,64,0,127,-89,0,112,15,-97,0,127,-47,0,127,-87,0,3,28,96,-69,0,2,3,-3,2,98,26,7,-91,0,64,32,16,12,4,-4,2,-3,1,-8,0,64,64,-5,32,-3,16,-3,8,-8,4,2,2,4,8,8,-5,16,-6,3,2,-6,16

780 DATA -3,32,-4,16,-7,8,16,32,32,64,-27,0,64,96,48,12,3,4,4,2,1,-104,0,-2,127,999

790 DATA -2,127,-100,0,3,4,4,-7,8,112,-67,0,1,31,32,64,64,-85,0,112,15,-97,0,124,3,-48,0,31,96,-88,0,15,112,-69,0,64,32,28,3,-28,0,-15,64,-15,32,-15,16,120,-14,8,-5,4,7,-4,4,-10,2,1,121,69,7,1,1,-61,0,1,1,2,2,4,4,8,8,16,16,32,32,64,64

800 DATA -4,0,64,64,-2,32,40,100,58,2,1,-112,0,-2,127,999

810 DATA -2,127,-111,0,1,2,-3,4,8,8,16,16,32,64,-54,0,64,32,32,-3,16,8,9,6,-82,0,112,15,-97,0,96,31,-49,0,127,-90,0,127,-62,0,96,16,-3,8,4,7,4,-14,2,-16,1,-3,0,127,-42,0,7,24,96,-34,0,-6,1,2,2,4,8,16,-3,32,64,-59,0,64,32,16,8,5,3,1,1,-122,0,-2,127,999

820 DATA -2,127,-121,0,1,6,24,32,64,-46,0,64,32,16,8,7,-89,0,112,15,-98,0,127,-48,0,48,46,65,64,-89,0,15,112,-59,0,112,15,-41,0,127,-44,0,3,28,96,-47,0,1,1,2,4,4,8,24,16,32,64,64,-43,0,64,32,32,16,15,-130,0,-2,127,999

830 DATA -2,127,-125,0,1,2,124,-42,0,30,33,64,-92,0,112,15,-98,0,112,15,-52,0,3,4,-5,2,-3,4,28,-6,32,-10,64,-63,0,127,-54,0,96,16,8,4,2,1,-43,0,127,-45,0,1,14,112,-55,0,1,1,2,2,4,8,8,16,32,64,-28,0,96,24,8,4,2,1,-135,0,-2,127,999

840 DATA -2,127,-128,0,3,56,68,-18,4,-18,8,48,76,4,2,1,-91,0,112,15,-100,0,127,-80,0,-5,1,-4,2,4,12,8,4,2,1,2,4,-3,8,4,4,-3,2,-3,4,8,8,16,16,8,-16,4,2,2,4,-5,8,-6,16,31,16,16,-4,32,64,-45,0,126,1,-49,0,127,-48,0,3,12,16,96,-61,0

```

850 DATA 1,1,2,4,8,16,16,-3,32,-4,64,32,64,72,76,116,8,16,8,4,-4,2,3,-141,0,-2,1
    27,999
860 DATA -2,127,-130,0,99,44,16,-28,0,124,2,65,62,-4,0,1,1,1,2,2,4,4,8,8,-3,16
    ,32,32,-3,64,-76,0,127,-100,0,112,15,-150,0,7,120,-35,64,-10,32,33,34,44,48,
    64,-45,0,127,-52,0,1,2,4,56,64,-66,0,32,112,80,19,4,-154,0,-2,127,999
870 DATA -2,127,-130,0,3,4,-3,8,48,64,-23,0,64,32,17,19,13,1,-3,2,14,16,-3,32,-1
    3,0,1,1,2,2,2,4,4,8,8,16,16,-3,32,-3,64,-56,0,64,63,-101,0,127,-152,0,127,
    -49,0,15,112,-44,0,127,-54,0,126,1,-65,0,14,9,114,1,-157,0,-2,127,999
880 DATA -2,127,-205,0,-3,1,2,2,-3,4,-3,8,-2,16,-3,32,-3,64,-36,0,112,15,-16,0,9
    6,-15,32,-8,64,112,-20,16,-20,32,-20,64,126,1,-152,0,127,-46,0,96,16,8,6,1,-
    45,0,127,-53,0,1,126,-65,0,126,1,3,1,-157,0,-2,127,999
890 DATA -2,127,-224,0,1,1,-13,2,-13,4,-8,8,15,-4,8,-12,16,28,3,-24,0,1,2,2,-3,4
    ,8,16,32,32,64,-203,0,3,4,8,24,112,-39,0,112,12,3,-50,0,127,-19,0,-12,64,-12
    ,32,-11,16,19,28,20,32,64,-3,0,-43,64,-3,0,112,-9,8,73,62,-161,0,-2,127,999
900 DATA -2,127,-200,0,-114,0,1,1,2,4,4,8,16,32,64,64,-198,0,3,4,4,8,8,112,-30,0
    ,16,24,23,-32,16,112,-20,0,31,96,-5,0,64,-9,0,112,14,1,-40,0,1,1,2,3,-42,0,1
    ,2,4,4,3,-8,0,2,3,1,2,12,48,64,-156,0,-2,127,999
910 DATA -2,127,-200,0,-124,0,1,2,2,4,4,8,112,-192,0,64,48,8,4,3,-63,0,4,11,8,8,
    16,32,64,64,32,16,16,-6,32,-3,16,32,32,16,16,32,31,16,16,8,15,9,66,108,88,88
    ,32,32,16,8,4,15,-4,4,5,2,5,-3,4,-12,2,12,-9,8,16,16,32,32,64,-18,0
920 DATA 96,-3,16,-3,8,-4,4,8,16,32,64,64,-36,0,3,4,24,32,64,-152,0,-2,127,999
930 DATA -2,127,-200,0,-130,0,7,120,-190,0,112,15,-7,0,96,32,64,-26,0,32,80,8,8,
    16,32,64,-27,0,56,36,35,16,8,8,16,96,-66,0,15,8,-4,4,12,-4,8,-4,4,2,1,-16,
    0,1,2,-4,4,-3,8,-3,16,32,32,64,64,-26,0,1,2,4,8,16,32,32,64,64,-143,0,-2,127
    ,999
940 DATA -2,127,-200,0,-131,0,3,2,4,8,16,16,32,64,-27,0,64,48,-4,8,12,4,2,-3,4,8
    ,8,-4,16,-4,32,64,64,-114,0,112,-2,16,112,-7,0,-3,64,-3,32,16,15,62,32,64,48
    ,-3,32,63,0,63,-5,16,60,83,82,87,-5,88,72,-3,64,-2,32,40,84,84,98,98,68,4,12
    ,17
950 DATA 34,66,68,69,42,90,28,-17,0,4,52,62,-4,16,96,0,0,12,50,34,66,66,1,-116,0
    ,1,126,-33,0,1,1,2,2,4,8,112,-136,0,-2,127,999
960 DATA -2,127,-200,0,-139,0,1,1,2,2,4,4,8,8,16,16,32,32,64,64,-3,0,64,64,-3,32
    ,-3,16,12,2,1,-24,0,1,1,2,4,8,16,32,64,-100,0,64,64,-3,32,39,40,17,5,11,9,5,
    4,2,1,1,-54,0,1,6,8,16,16,8,56,48,32,16,24,-5,4,2,12,16,16,24,7,-4,0
970 DATA 1,-3,2,-3,4,21,41,34,34,58,10,10,4,-109,0,95,32,-38,0,3,12,48,64,-133,0
    ,-2,127,999
980 DATA -2,127,-200,0,-153,0,1,1,1,-43,0,1,2,4,8,16,32,64,-66,0,-5,64,-12,0,64,
    64,32,16,16,8,4,2,2,1,-220,0,31,96,-42,0,1,2,4,8,16,32,64,-126,0,-2,127,999
990 DATA -2,127,-200,0,-206,0,3,12,48,64,-56,0,-4,64,0,112,83,36,40,24,16,9,9,-3
    ,5,3,3,3,2,1,1,1,-231,0,1,1,2,2,4,4,8,16,32,64,-5,0,64,64,0,64,64,32,96,-28,
    0,3,4,8,48,64,-121,0,-2,127,999
1000 DATA -2,127,-200,0,-209,0,1,2,4,8,16,32,64,64,-41,0,16,24,40,80,80,22,85,53
    ,8,8,0,4,3,1,-200,0,-58,0,1,2,4,8,4,3,16,40,81,72,16,17,46,80,32,64,-28,0,1
    27,-121,0,-2,127,999
1010 DATA -2,127,-200,0,-217,0,1,2,2,62,64,-31,0,8,20,20,36,100,2,66,50,13,3,-20
    0,0,-77,0,1,2,4,8,17,33,64,-26,0,64,39,24,-120,0,-2,127,999
1020 DATA -2,127,-200,0,-221,0,1,6,12,16,32,64,64,-29,0,48,79,127,-200,0,-87,0,-
    6,1,2,-3,4,8,40,80,64,64,-9,0,124,2,1,-122,0,-2,127,999
1030 DATA -2,127,-200,0,-228,0,1,2,4,8,16,16,32,32,-3,64,-19,0,3,4,27,100,24,96,
    -200,0,-96,0,4,10,11,-3,4,-3,66,34,17,17,13,7,2,-121,0,-2,127,999
1040 DATA -2,127,-200,0,-239,0,1,1,2,2,-6,4,-6,8,-3,16,32,16,16,8,9,85,74,32,-20
    0,0,-84,0,4,-5,2,-9,1,-131,0,-2,127,999
1050 DATA -2,127,-200,0,-200,0,-200,0,-200,0,-96,0,-2,127,999

```


APPENDIX B

Code Sequences for Fourteen Custom Alphabets

PROGRAM B-1. ALPHABET. General program for storage of code sequences.

```
10 'PROGRAM B-1: "ALPHABET" -- GENERAL PROGRAM FOR STORING ALPHABET STRINGS
20 CLEAR 4000
30 DIM L$(40),C(13)
40 INPUT "POSITION (0 TO 480)";P
50 P1=FIX(P/256): P2=FIX(P-256*P1)
60 PN$=CHR$(27)+CHR$(16)+CHR$(P1)+CHR$(P2)
70 CD$=CHR$(30)+CHR$(27)+CHR$(20)+CHR$(18)
80 NM$=CHR$(30)+CHR$(27)+CHR$(19)
90 LF$=CHR$(30)+CHR$(13)
100 FOR X=1 TO 36
110   PRINT X: PRINT
120   READ CHAR$
130   PRINT CHAR$;: LPRINT CHAR$;
140   FOR N=0 TO 12: READ C(N): C(N)=128+C(N): PRINT C(N);: LPRINT C(N);: NEXT N
150   L$(X)=CD$+CHR$(C(0))+CHR$(C(1))+CHR$(C(2))+CHR$(C(3))+CHR$(C(4))+CHR$(C(5))
      +CHR$(C(6))+CHR$(C(7))+CHR$(C(8))+CHR$(C(9))+CHR$(C(10))+CHR$(C(11))+CHR$
      (C(12))+NM$
160   A$=L$(1):B$=L$(2):C$=L$(3):D$=L$(4):E$=L$(5):F$=L$(6):G$=L$(7):H$=L$(8):I$
      =L$(9):J$=L$(10):K$=L$(11):L$=L$(12):M$=L$(13)
170   N$=L$(14):O$=L$(15):P$=L$(16):Q$=L$(17):R$=L$(18):S$=L$(19):T$=L$(20):U$=L
      $(21):V$=L$(22):W$=L$(23):X$=L$(24):Y$=L$(25):Z$=L$(26)
180   N0$=L$(27):N1$=L$(28):N2$=L$(29):N3$=L$(30):N4$=L$(31):N5$=L$(32):N6$=L$(3
      3):N7$=L$(34):N8$=L$(35):N9$=L$(36)
190   LPRINT PN$;L$(X);LF$;
200 NEXT X
210 LPRINT A$B$C$D$E$F$G$H$I$J$K$L$M$N$O$P$Q$R$S$T$U$V$W$X$Y$Z$N0$N1$N2$N3$N4$N5
      $N6$N7$N8$N9$
220 END
1000 'DATA -- DOT CODES IN BINARY-SUM FORM (0 TO 127 FOR TRS-80, 0 TO 255 FOR IB
      M-EPSON AND APPLE)
```

PROGRAM B-2. PERP5X5/DAT. Perpendicular five-by-five capitals and numbers.

```

10 'PROGRAM B-2: "PERP5X5/DAT" -- BINARY-SUM CODES FOR PERPENDICULAR 5X5 SET
20 'FOR TRS-80 PRINTERS
1000 REM -- CODE SEQUENCES FOR LETTERS (1001-1026) AND NUMBERS (1030-1039)
1001 DATA 0,34,62,34,20,8,0
1002 DATA 0,30,34,30,34,30,0
1003 DATA 0,28,34,2,34,28,0
1004 DATA 0,30,34,34,34,30,0
1005 DATA 0,62,2,14,2,62,0
1006 DATA 0,2,2,14,2,62,0
1007 DATA 0,28,34,58,2,28,0
1008 DATA 0,34,34,62,34,34,0
1009 DATA 0,28,8,8,8,28,0
1010 DATA 0,12,18,16,16,56,0
1011 DATA 0,34,18,14,18,34,0
1012 DATA 0,62,2,2,2,2,0
1013 DATA 0,34,42,42,54,34,0
1014 DATA 0,34,50,42,38,34,0
1015 DATA 0,28,34,34,34,28,0
1016 DATA 0,2,2,30,34,30,0
1017 DATA 0,60,50,42,34,28,0
1018 DATA 0,34,18,30,34,30,0
1019 DATA 0,28,32,28,2,28,0
1020 DATA 0,8,8,8,8,62,0
1021 DATA 0,28,34,34,34,34,0
1022 DATA 0,8,20,34,34,34,0
1023 DATA 0,34,54,42,42,34,0
1024 DATA 0,34,20,8,20,34,0
1025 DATA 0,8,8,8,20,34,0
1026 DATA 0,62,4,8,16,62,0
1030 DATA 0,28,34,34,34,28,0
1031 DATA 0,28,8,8,8,12,0
1032 DATA 0,62,4,24,34,28,0
1033 DATA 0,28,32,24,32,28,0
1034 DATA 0,16,62,20,24,16,0
1035 DATA 0,30,32,30,2,62,0
1036 DATA 0,28,34,26,2,28,0
1037 DATA 0,4,4,8,16,62,0
1038 DATA 0,28,34,28,34,28,0
1039 DATA 0,28,32,44,34,28,0

```

PROGRAM B-3. UPRT5X7/DAT. Upright five-by-seven capital letters.

```

10 'PROGRAM B-3: "UPRT5X7/DAT" -- UPRIGHT 5X7 CAPITAL LETTERS (TRS-80)
1000 REM -- CODES FOR CAPITAL LETTERS (1001-1026)
1001 DATA A,0,124,10,9,10,124,0
1002 DATA B,0,127,73,73,73,54,0
1003 DATA C,0,62,65,65,65,34,0
1004 DATA D,0,127,65,65,65,62,0

```

```

1005 DATA E,0,127,73,73,65,65,0
1006 DATA F,0,127,9,9,1,1,0
1007 DATA G,0,62,65,73,73,58,0
1008 DATA H,0,127,8,8,8,127,0
1009 DATA I,0,0,65,127,65,0,0
1010 DATA J,0,32,64,65,63,1,0
1011 DATA K,0,127,8,20,34,65,0
1012 DATA L,0,127,64,64,64,64,0
1013 DATA M,0,127,2,28,2,127,0
1014 DATA N,0,127,4,8,16,127,0
1015 DATA O,0,62,65,65,65,62,0
1016 DATA P,0,127,9,9,9,6,0
1017 DATA Q,0,30,33,49,33,94,0
1018 DATA R,0,127,9,25,41,70,0
1019 DATA S,0,38,73,73,73,50,0
1020 DATA T,0,1,1,127,1,1,0
1021 DATA U,0,63,64,64,64,63,0
1022 DATA V,0,3,28,96,28,3,0
1023 DATA W,0,63,64,56,64,63,0
1024 DATA X,0,99,20,8,20,99,0
1025 DATA Y,0,3,4,120,4,3,0
1026 DATA Z,0,97,81,73,69,67,0
1027 DATA -,0,0,8,8,8,0,0
1028 DATA +,0,8,8,62,8,8,0
1099 DATA ZZZ,0,0,0,0,0,0,0

```

PROGRAM B-4. PERP5X7/DAT. Perpendicular five-by-seven capital letters.

```

1000 'PROGRAM B-4: "PERP5X7/DAT" -- PERPENDICULAR 5X7 CAPITALS & PUNCTUATION
1001 DATA 0,34,34,62,34,54,20,28,0
1002 DATA 0,30,34,34,30,34,34,30,0
1003 DATA 0,28,34,2,2,2,34,28,0
1004 DATA 0,30,34,34,34,34,34,30,0
1005 DATA 0,62,2,2,14,2,2,62,0
1006 DATA 0,2,2,2,14,2,2,62,0
1007 DATA 0,28,34,34,58,2,34,28,0
1008 DATA 0,34,34,34,62,34,34,34,0
1009 DATA 0,28,8,8,8,8,8,28,0
1010 DATA 0,12,18,16,16,16,16,56,0
1011 DATA 0,34,18,10,6,10,18,34,0
1012 DATA 0,62,2,2,2,2,2,2,0
1013 DATA 0,34,34,34,42,42,54,34,0
1014 DATA 0,34,34,50,42,38,34,34,0
1015 DATA 0,28,34,34,34,34,34,28,0
1016 DATA 0,2,2,2,30,34,34,30,0
1017 DATA 0,32,28,42,34,34,34,28,0
1018 DATA 0,34,18,10,30,34,34,30,0
1019 DATA 0,28,34,32,28,2,34,28,0
1020 DATA 0,8,8,8,8,8,8,62,0
1021 DATA 0,28,34,34,34,34,34,34,0
1022 DATA 0,8,20,20,34,34,34,34,0
1023 DATA 0,20,42,42,42,34,34,0

```

```

1024 DATA 0,34,34,20,8,20,34,34,0
1025 DATA 0,8,8,8,8,20,34,34,0
1026 DATA 0,62,2,4,8,16,32,62,0
1027 DATA 0,0,0,0,28,0,0,0
1028 DATA 0,0,2,4,8,16,32,0,0
1029 DATA 0,0,12,12,0,12,12,0,0
1030 DATA 0,0,8,8,62,8,8,0,0

```

PROGRAM B-5. UPDN5X7/DAT. Upside-down five-by-seven capital letters.

```

10 'PROGRAM B-5: "UPDN5X7/DAT" -- UPSIDE-DOWN 5X7 CAPITAL LETTERS (TRS-80)
1000 REM -- CODES FOR CAPITAL LETTERS (1001-1026)
1001 DATA A,0,31,36,68,36,31,0
1002 DATA B,0,54,73,73,73,127,0
1003 DATA C,0,34,65,65,65,62,0
1004 DATA D,0,62,65,65,65,127,0
1005 DATA E,0,65,65,73,73,127,0
1006 DATA F,0,64,64,72,72,127,0
1007 DATA G,0,46,73,73,65,62,0
1008 DATA H,0,127,8,8,8,127,0
1009 DATA I,0,0,65,127,65,0,0
1010 DATA J,0,64,126,65,1,2,0
1011 DATA K,0,65,34,20,8,127,0
1012 DATA L,0,1,1,1,1,127,0
1013 DATA M,0,127,32,24,32,127,0
1014 DATA N,0,127,4,8,16,127,0
1015 DATA O,0,62,65,65,65,62,0
1016 DATA P,0,48,72,72,72,127,0
1017 DATA Q,0,61,66,70,66,60,0
1018 DATA R,0,49,74,76,72,127,0
1019 DATA S,0,38,73,73,73,49,0
1020 DATA T,0,64,64,127,64,64,0
1021 DATA U,0,126,1,1,1,126,0
1022 DATA V,0,96,28,3,28,96,0
1023 DATA W,0,126,1,14,1,126,0
1024 DATA X,0,99,20,8,20,99,0
1025 DATA Y,0,96,16,15,16,96,0
1026 DATA Z,0,97,81,73,69,67,0
1027 DATA -,0,0,8,8,8,0,0
1028 DATA +,0,8,8,62,8,8,0
1099 DATA ZZZ,0,0,0,0,0,0,0

```

PROGRAM B-6. PERP5X11/DAT. Perpendicular five-by-eleven condensed characters.

```

10 'PROGRAM B-6: "PERP5X11/DAT" -- PERPENDICULAR 5 X 11 CONDENSED (TRS-80)
1000 REM -- CODE SEQUENCES FOR LETTERS (1001-1026) AND NUMBERS (1027-1036)
1001 DATA A,0,34,34,34,34,62,34,34,34,20,28,8,0
1002 DATA B,0,30,18,34,34,18,30,18,34,34,18,30,0
1003 DATA C,0,28,20,34,34,2,2,2,34,34,20,28,0
1004 DATA D,0,30,18,34,34,34,34,34,34,18,30,0
1005 DATA E,0,62,34,34,2,2,14,2,2,34,34,62,0

```

```

1006 DATA F,0,2,2,2,2,2,14,2,2,34,34,62,0
1007 DATA G,0,60,34,34,34,42,58,2,2,34,34,28,0
1008 DATA H,0,34,34,34,34,34,34,62,34,34,34,34,0
1009 DATA I,0,28,8,8,8,8,8,8,8,8,8,28,0
1010 DATA J,0,12,18,18,16,16,16,16,16,16,16,56,0
1011 DATA K,0,34,34,18,10,6,2,6,10,18,34,34,0
1012 DATA L,0,30,2,2,2,2,2,2,2,2,2,0
1013 DATA M,0,34,34,34,34,42,42,42,54,54,34,34,0
1014 DATA N,0,34,34,50,50,58,42,46,38,38,34,34,0
1015 DATA O,0,28,20,34,34,34,34,34,34,34,20,28,0
1016 DATA P,0,2,2,2,2,2,30,34,34,34,34,30,0
1017 DATA Q,0,32,28,50,42,34,34,34,34,34,20,28,0
1018 DATA R,0,34,34,18,10,6,30,34,34,34,18,30,0
1019 DATA S,0,28,34,34,32,32,28,2,2,34,34,28,0
1020 DATA T,0,8,8,8,8,8,8,8,8,42,42,62,0
1021 DATA U,0,28,34,34,34,34,34,34,34,34,34,0
1022 DATA V,0,8,8,28,20,20,20,34,34,34,34,0
1023 DATA W,0,34,34,54,54,54,42,42,42,34,34,34,0
1024 DATA X,0,34,34,20,20,8,8,8,20,20,34,34,0
1025 DATA Y,0,8,8,8,8,8,8,20,20,34,34,0
1026 DATA Z,0,62,2,4,4,8,8,8,16,16,32,62,0
1027 DATA 0,0,28,34,34,34,34,34,34,34,34,28,0
1028 DATA 1,0,28,8,8,8,8,8,8,8,12,8,8,0
1029 DATA 2,0,62,2,2,4,8,16,32,32,34,34,28,0
1030 DATA 3,0,28,34,34,32,16,28,16,32,32,34,28,0
1031 DATA 4,0,16,16,16,16,62,18,20,24,16,16,16,0
1032 DATA 5,0,28,34,34,32,32,32,30,2,2,2,62,0
1033 DATA 6,0,28,34,34,34,34,30,2,2,34,34,28,0
1034 DATA 7,0,4,4,4,8,8,16,16,32,32,34,62,0
1035 DATA 8,0,28,34,34,34,34,28,20,34,34,34,28,0
1036 DATA 9,0,28,34,34,32,32,60,34,34,34,34,28,0

```

PROGRAM B-7. NUMS5X11/STR. Perpendicular five-by-eleven numbers stored in strings.

```

10 'PROGRAM B-7: "NUMS5X11/STR" -- PERP. 5 X 11 COND. NUMBER STRINGS (TRS-80)
1000 'CODE STRINGS FOR NUMBERS 1-9, 0, AND HYPHEN
1010 CD$=CHR$(30)+CHR$(27)+CHR$(20)+CHR$(18): NM$=CHR$(30)+CHR$(27)+CHR$(19)+CHR$(18)
1020 N1$=CD$+CHR$(156)+STRING$(7,136)+CHR$(140)+STRING$(2,136)+NM$
1030 N2$=CD$+CHR$(190)+STRING$(2,130)+CHR$(132)+CHR$(136)+CHR$(144)+STRING$(2,160)+STRING$(2,162)+CHR$(156)+NM$
1040 N3$=CD$+CHR$(156)+STRING$(2,162)+CHR$(160)+CHR$(144)+CHR$(156)+CHR$(144)+CHR$(160)+STRING$(2,162)+CHR$(156)+NM$
1050 N4$=CD$+STRING$(4,144)+CHR$(190)+CHR$(146)+CHR$(148)+CHR$(152)+STRING$(3,144)+NM$
1060 N5$=CD$+CHR$(156)+STRING$(2,162)+STRING$(3,160)+CHR$(158)+STRING$(3,130)+CHR$(190)+NM$
1070 N6$=CD$+CHR$(156)+STRING$(4,162)+CHR$(158)+STRING$(2,130)+STRING$(2,162)+CHR$(156)+NM$
1080 N7$=CD$+STRING$(3,132)+STRING$(2,136)+STRING$(2,144)+STRING$(2,160)+CHR$(162)+CHR$(190)+NM$

```

```

1090 N8$=CD$+CHR$(156)+STRING$(3,162)+CHR$(148)+CHR$(156)+CHR$(148)+STRING$(3,16
2)+CHR$(156)+NM$
1100 N9$=CD$+CHR$(156)+STRING$(2,162)+STRING$(2,160)+CHR$(188)+STRING$(4,162)+CH
R$(156)+NM$
1110 N0$=CD$+CHR$(156)+STRING$(9,162)+CHR$(156)+NM$
1120 HM$=CD$+STRING$(4,128)+STRING$(3,190)+STRING$(4,128)+NM$
1130 RETURN

```

PROGRAM B-8. UPRITAL/DAT. Upright italic full character set.

```

10 'PROGRAM B-8: "UPRTITAL/DAT" -- UPRIGHT 7 X 9 CONDENSED ITALICS (TRS-80)
1000 '***** CODE SEQUENCES FOR CAPITAL LETTERS *****
1001 DATA 96,24,20,18,17,17,81,50,12
1002 DATA 64,113,77,75,73,73,41,53,2
1003 DATA 48,76,66,66,65,65,33,3,0
1004 DATA 64,113,77,67,65,65,33,25,6
1005 DATA 96,88,78,73,73,73,1,1,0
1006 DATA 96,24,14,9,9,9,1,1,0
1007 DATA 48,76,66,66,73,73,41,25,10
1008 DATA 96,24,14,9,8,72,56,12,3
1009 DATA 0,64,64,97,89,71,1,1,0
1010 DATA 48,64,64,64,65,49,13,3,1
1011 DATA 96,24,14,9,8,20,100,2,1
1012 DATA 96,88,70,65,64,64,64,0,0
1013 DATA 96,28,3,4,24,4,98,28,3
1014 DATA 96,28,3,4,8,16,96,28,3
1015 DATA 48,76,66,66,65,65,33,25,6
1016 DATA 96,24,14,9,9,9,9,9,6
1017 DATA 48,76,66,66,65,81,33,89,6
1018 DATA 96,24,14,9,25,41,41,73,6
1019 DATA 32,70,73,73,73,73,73,49,2
1020 DATA 0,0,1,97,25,7,1,1,1
1021 DATA 112,76,67,64,64,64,48,12,3
1022 DATA 8,22,33,64,64,32,16,12,3
1023 DATA 96,60,35,16,12,16,96,28,3
1024 DATA 64,32,17,10,4,12,114,1,0
1025 DATA 0,0,67,36,24,8,4,2,1
1026 DATA 64,96,97,81,73,69,69,3,1
1027 DATA 0,64,0,8,4,2,1,0,0
1028 DATA 0,4,2,1,0,4,2,1,0
1029 DATA 84,52,28,22,85,52,28,22,21
1030 DATA 0,36,106,58,42,46,43,18,0
2000 '***** CODES FOR LOWER-CASE LETTERS *****
2001 DATA 32,112,80,84,84,84,116,84,8
2002 DATA 96,88,70,69,68,68,36,36,24
2003 DATA 48,72,72,68,68,68,68,4,0
2004 DATA 48,72,68,68,68,68,52,12,3
2005 DATA 48,88,88,84,84,84,20,20,8
2006 DATA 0,0,68,52,12,6,5,1,1
2007 DATA 76,82,82,82,82,50,26,12,0
2008 DATA 96,24,6,5,4,4,68,36,24
2009 DATA 0,64,64,100,84,76,1,0,0
2010 DATA 0,32,64,64,64,32,20,28,5

```

```

2011 DATA 0,96,24,22,17,48,72,8,4
2012 DATA 64,64,96,89,71,1,0,0,0
2013 DATA 64,36,28,68,36,28,68,36,24
2014 DATA 100,28,4,4,4,68,36,24,0
2015 DATA 48,72,68,68,68,68,36,24,0
2016 DATA 64,48,28,20,20,20,20,12,0
2017 DATA 0,8,20,82,82,114,82,12,0
2018 DATA 64,32,28,16,8,4,4,4,8
2019 DATA 64,64,72,84,84,84,36,4,4
2020 DATA 0,4,52,76,70,69,36,4,0
2021 DATA 0,48,72,68,64,64,96,88,4
2022 DATA 0,24,100,32,32,16,16,8,4
2023 DATA 96,60,32,16,8,48,64,32,28
2024 DATA 64,64,36,24,16,48,72,8,4
2025 DATA 0,0,76,80,32,16,8,4,0
2026 DATA 64,64,100,84,84,84,76,4,4
2027 DATA 0,66,35,19,73,100,98,33,0
2028 DATA 48,72,74,69,77,85,34,80,8
2029 DATA 0,0,4,5,2,1,0,0,0
2030 DATA 8,40,24,10,28,40,12,10,8
3000 '***** CODES FOR PUNCTUATION AND NUMBERS *****
3001 DATA 0,64,64,96,82,74,6,1,0
3002 DATA 64,68,98,80,81,73,73,5,2
3003 DATA 32,64,65,73,73,77,49,3,1
3004 DATA 0,16,24,16,84,48,26,20,3
3005 DATA 0,32,68,70,69,69,69,37,25
3006 DATA 48,72,68,68,74,74,73,40,16
3007 DATA 65,33,17,9,9,5,5,3,1
3008 DATA 32,84,74,74,73,9,41,21,2
3009 DATA 0,4,74,42,41,9,17,9,6
3010 DATA 92,34,97,81,73,69,67,34,29
3011 DATA 0,0,24,36,66,1,1,0,0
3012 DATA 0,0,64,64,33,18,12,0,0
3013 DATA 8,8,40,24,8,12,10,8,8
3014 DATA 0,0,64,80,32,16,0,0,0
3015 DATA 0,8,8,8,8,8,8,0,0
3016 DATA 0,64,96,96,32,0,0,0,0
3017 DATA 0,64,32,16,8,4,2,1,0
3018 DATA 96,88,70,65,65,1,1,1,0
3019 DATA 0,64,64,64,65,65,49,13,3
3020 DATA 0,0,1,2,12,48,64,0,0
3021 DATA 4,0,2,0,1,2,4,0,0
3022 DATA 64,64,64,64,64,64,64,64,0
3023 DATA 0,0,0,0,7,1,1,0,0

```

PROGRAM B-9. PERP7X10/DAT. Perpendicular bold seven-by-ten capitals and numbers.

```

10 'PROGRAM B-9: "PERP7X10/DAT" -- PERPENDICULAR BOLD 7 X 10 ALPHABET (TRS-80)
1000 REM -- CODES FOR CAPITALS (1001-1026)
1001 DATA A,0,99,99,99,99,127,127,99,99,62,28,0
1002 DATA B,0,63,127,99,99,63,63,99,99,127,63,0
1003 DATA C,0,62,127,99,3,3,3,3,99,127,62,0

```

```

1004 DATA D,0,63,127,99,99,99,99,99,99,127,63,0
1005 DATA E,0,127,127,3,3,31,31,3,3,127,127,0
1006 DATA F,0,3,3,3,3,31,31,3,3,127,127,0
1007 DATA G,0,62,127,99,99,123,123,3,99,127,62,0
1008 DATA H,0,99,99,99,99,127,127,99,99,99,99,0
1009 DATA I,0,60,60,24,24,24,24,24,24,60,60,0
1010 DATA J,0,30,63,51,51,48,48,48,48,120,120,0
1011 DATA K,0,99,51,27,15,7,7,15,27,51,99,0
1012 DATA L,0,127,127,3,3,3,3,3,3,3,3,0
1013 DATA M,0,99,99,99,99,107,107,127,119,99,99,0
1014 DATA N,0,99,99,99,115,123,111,103,99,99,99,0
1015 DATA O,0,62,127,99,99,99,99,99,99,127,62,0
1016 DATA P,0,3,3,3,3,63,127,99,99,127,63,0
1017 DATA Q,0,16,62,63,123,99,99,99,99,127,62,0
1018 DATA R,0,99,99,99,99,63,63,99,99,127,63,0
1019 DATA S,0,62,127,99,96,126,63,3,99,127,62,0
1020 DATA T,0,24,24,24,24,24,24,24,24,126,126,0
1021 DATA U,0,62,127,99,99,99,99,99,99,99,99,0
1022 DATA V,0,8,28,54,54,54,54,99,99,99,99,0
1023 DATA W,0,99,99,119,127,107,107,99,99,99,99,0
1024 DATA X,0,99,99,119,62,28,28,62,119,99,99,0
1025 DATA Y,0,24,24,24,24,60,126,102,102,102,102,0
1026 DATA Z,0,127,127,3,6,12,24,48,96,127,127,0
1030 DATA 0,0,62,127,99,99,99,99,99,99,127,62,0
1031 DATA 1,0,30,30,12,12,12,12,12,14,12,8,0
1032 DATA 2,0,127,127,3,6,60,120,96,99,127,62,0
1033 DATA 3,0,62,127,99,96,60,60,96,99,127,62,0
1034 DATA 4,0,48,48,48,127,127,51,54,60,56,48,0
1035 DATA 5,0,62,127,99,96,96,63,31,3,127,127,0
1036 DATA 6,0,62,127,99,99,127,63,3,99,127,62,0
1037 DATA 7,0,6,6,6,12,24,48,99,99,127,127,0
1038 DATA 8,0,62,127,99,99,62,62,99,99,127,62,0
1039 DATA 9,0,62,127,99,96,126,127,99,99,127,62,0
1099 DATA ZZZ

```

PROGRAM B-10. PERPSERF/DAT. Perpendicular seven-by-eleven serif capitals and numbers.

```

10 'PROGRAM B-10: "PERPSERF/DAT" -- PERPEND. 7 X 11 SERIF ALPHABET (TRS-80)
1000 REM -- CODE SEQUENCES FOR LETTERS (1001-1026) AND NUMBERS (1030-1039)
1001 DATA 0,119,34,34,34,62,34,34,34,20,20,28,0
1002 DATA 0,63,66,66,66,34,30,34,66,66,66,63,0
1003 DATA 0,28,34,65,65,1,1,1,65,65,98,92,0
1004 DATA 0,31,34,66,66,66,66,66,66,34,31,0
1005 DATA 0,127,66,66,2,18,30,18,2,66,66,127,0
1006 DATA 0,7,2,2,2,18,30,18,2,66,66,127,0
1007 DATA 0,46,49,33,33,121,1,1,33,33,49,46,0
1008 DATA 0,119,34,34,34,34,62,34,34,34,34,119,0
1009 DATA 0,28,8,8,8,8,8,8,8,8,28,0
1010 DATA 0,12,18,33,33,33,32,32,32,32,112,0
1011 DATA 0,119,34,18,10,6,2,6,10,18,34,119,0
1012 DATA 0,127,66,66,2,2,2,2,2,2,2,7,0
1013 DATA 0,99,65,65,65,73,85,85,99,99,65,0

```



```

1014 DATA 0,119,34,50,50,42,42,42,38,38,34,119,0
1015 DATA 0,28,34,65,65,65,65,65,65,65,34,28,0
1016 DATA 0,7,2,2,2,2,30,34,66,66,34,31,0
1017 DATA 0,64,60,54,73,65,65,65,65,65,34,28,0
1018 DATA 0,119,34,34,34,18,30,34,66,66,34,31,0
1019 DATA 0,29,35,65,65,32,28,2,65,65,98,92,0
1020 DATA 0,28,8,8,8,8,8,8,8,73,73,127,0
1021 DATA 0,28,34,34,34,34,34,34,34,34,119,0
1022 DATA 0,8,8,28,20,20,54,34,34,34,34,119,0
1023 DATA 0,65,99,99,85,85,73,73,65,65,65,99,0
1024 DATA 0,119,34,34,20,20,8,20,20,34,34,119,0
1025 DATA 0,28,8,8,8,8,8,28,20,34,34,119,0
1026 DATA 0,127,65,65,2,4,8,16,32,65,65,127,0
1030 DATA 0,28,34,34,34,34,34,34,34,34,28,0
1031 DATA 0,28,8,8,8,8,8,8,8,12,8,8,0
1032 DATA 0,127,65,66,4,8,16,32,65,65,34,28,0
1033 DATA 0,28,34,65,65,64,32,24,16,33,65,127,0
1034 DATA 0,112,32,32,32,127,33,34,36,40,48,32,0
1035 DATA 0,28,34,65,64,64,32,31,1,65,65,127,0
1036 DATA 0,28,34,65,65,35,29,1,65,65,34,28,0
1037 DATA 0,4,4,4,4,8,8,8,16,17,33,127,0
1038 DATA 0,28,34,65,65,34,28,34,65,65,34,28,0
1039 DATA 0,28,34,65,65,64,92,98,65,65,34,28,0
1040 DATA 0,55,74,74,74,74,74,74,74,75,74,50,0
1041 DATA 0,119,34,34,34,34,34,34,34,51,34,34,0
1042 DATA 0,122,74,74,18,18,34,34,66,67,74,50,0

```

PROGRAM B-11. PERP716S/DAT. Perpendicular seven-by-sixteen serif capitals—Epson.

```

1000 'PROGRAM B-11: "PERP716S/DAT" -- PERPEND. 7 X 16 SERIF ALPHABET (EPSON)
1001 DATA 0,0,238,238,68,68,68,68,124,124,68,68,68,68,40,40,56,16,0,0
1002 DATA 0,0,248,252,68,68,68,68,68,120,120,68,68,68,68,252,248,0,0
1003 DATA 0,0,56,124,68,68,68,64,64,64,64,64,64,68,68,68,124,60,0,0
1004 DATA 0,0,248,252,68,68,68,68,68,68,68,68,68,68,68,68,68,68,252,248,0,0
1005 DATA 0,0,252,252,68,68,64,64,72,120,120,72,64,64,68,68,252,252,0,0
1006 DATA 0,0,224,224,64,64,64,64,72,120,120,72,64,64,68,68,252,252,0,0
1007 DATA 0,0,60,124,68,68,68,94,94,64,64,64,64,68,68,76,124,52,0,0
1008 DATA 0,0,238,238,68,68,68,68,68,124,124,68,68,68,68,238,238,0,0
1009 DATA 0,0,56,56,16,16,16,16,16,16,16,16,16,16,16,16,56,56,0,0
1010 DATA 0,0,56,124,68,68,68,228,228,4,4,4,4,4,4,4,14,14,0,0
1011 DATA 0,0,238,238,68,68,76,88,112,96,96,112,88,76,68,68,238,238,0,0
1012 DATA 0,0,252,252,68,68,64,64,64,64,64,64,64,64,64,64,224,224,0,0
1013 DATA 0,0,238,238,68,68,68,68,68,84,84,84,124,108,108,68,68,68,0,0
1014 DATA 0,0,230,230,76,76,76,76,84,84,84,84,100,100,100,100,206,206,0,0
1015 DATA 0,0,56,124,68,68,68,68,68,68,68,68,68,68,68,68,124,56,0,0
1016 DATA 0,0,224,224,64,64,64,64,112,120,76,68,68,68,76,124,248,0,0
1017 DATA 0,0,2,60,124,76,116,68,68,68,68,68,68,68,68,124,56,0,0
1018 DATA 0,0,238,238,68,68,68,68,72,120,120,72,68,68,68,68,124,248,0,0
1019 DATA 0,0,120,124,68,68,4,4,4,60,120,64,64,64,68,68,124,60,0,0
1020 DATA 0,0,56,56,16,16,16,16,16,16,16,16,16,16,84,84,124,124,0,0
1021 DATA 0,0,56,124,68,68,68,68,68,68,68,68,68,68,68,68,238,238,0,0
1022 DATA 0,0,16,16,56,40,40,40,68,68,68,68,68,68,68,68,238,238,0,0
1023 DATA 0,0,40,40,108,124,84,84,84,84,84,68,68,68,68,68,238,238,0,0

```

1026 DATA 0,0,254,254,130,66,66,32,32,16,16,8,8,4,132,130,254,254,0,0

PROGRAM B-12. PERP8X18/DAT. Perpendicular eight-by-eighteen bold capitals and numbers —Epson.

```

10 'PROGRAM B-12: "PERPBX18/DAT" -- BOLD PERPEND. 8 X 18 (EPSON DOUBLE DENSITY)
1000 'EPSON CODES FOR CAPITAL LETTERS (1001-1026) & NUMBERS (1030-1040)
1001 DATA 195,195,195,195,195,195,255,255,255,195,195,195,195,231,102,126, 60, 60
1002 DATA 252,254,255,199,195,195,199,206,252,252,206,199,195,195,199,255,254,252
1003 DATA 60,126,127,231,195,195,192,192,192,192,192,192,195,195,231,127,126, 60
1004 DATA 252,254,255,199,195,195,195,195,195,195,195,195,195,199,255,254,252
1005 DATA 255,255,255,192,192,192,192,192,255,255,255,192,192,192,192,255,255,255
1006 DATA 192,192,192,192,192,192,192,192,255,255,255,192,192,192,192,255,255,255
1007 DATA 63,127,127,227,195,195,207,207,207,192,192,192,195,195,231,127,126, 60
1008 DATA 195,195,195,195,195,195,195,255,255,255,195,195,195,195,195,195,195,195
1009 DATA 126,126,126, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,126,126,126
1010 DATA 60,126,255,231,195,195,195, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
1011 DATA 195,199,198,206,204,220,216,240,240,240,240,216,220,204,206,198,199,195
1012 DATA 255,255,255,192,192,192,192,192,192,192,192,192,192,192,192,192,192,192
1013 DATA 195,195,195,195,195,195,195,195,235,235,235,255,255,231,231,231,195,195
1014 DATA 195,195,199,199,199,207,207,203,219,219,211,243,243,227,227,227,185,185
1015 DATA 60,126,126,231,195,195,195,195,195,195,195,195,195,195,231,126,126, 60
1016 DATA 192,192,192,192,192,192,192,192,252,254,255,199,195,195,199,255,254,252
1017 DATA 6, 6, 62,126,255,255,219,195,195,195,195,195,195,195,231,126,126, 60
1018 DATA 195,195,195,195,199,206,220,248,252,254,255,199,195,195,199,255,254,252
1019 DATA 60,126,254,199,195, 3, 3, 7, 63,126,252,224,192,195,199,127,126, 60
1020 DATA 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,255,255,255
1021 DATA 60,126,255,231,195,195,195,195,195,195,195,195,195,195,195,195,195,195
1022 DATA 24, 24, 24, 60, 60, 60,126,102,102,102,102,195,195,195,195,195,195,195
1023 DATA 102,102,102,231,231,255,255,219,219,219,195,195,195,195,195,195,195,195
1024 DATA 195,195,195,102,102, 60, 60, 24, 24, 24, 24, 60, 60,102,102,195,195,195
1025 DATA 24, 24, 24, 24, 24, 24, 24, 24, 60, 60,102,102,102,195,195,195,195,195
1026 DATA 255,255,255,192,192, 96, 96, 48, 56, 28, 12, 8, 8 3, 3,255,255,255
1030 DATA 60,126,255,231,195,195,195,195,195,195,195,195,195,195,231,255,126, 60
1031 DATA 60, 60, 60, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24,120,56, 24, 8
1032 DATA 255,255,255,192,224,112, 46, 24, 12, 6, 7, 3,195,195,231,255,126, 60
1033 DATA 60,126,255,231,195,195, 7, 14, 60, 60, 14, 7,195,195,231,255,126, 60
1034 DATA 6, 6, 6, 6, 6, 6,255,255,255,198,102,102, 54, 56, 24, 28, 12, 14
1035 DATA 60,126,255,231,195,195, 3, 3, 7,255,254,252,195,195,195,255,255,255
1036 DATA 60,126,255,231,195,195,231,223,222,220,192,192,195,195,231,255,126, 60
1037 DATA 48, 48, 48, 48, 48, 48, 24, 24, 24, 12, 12, 6, 6, 3, 3,255,255,255
1038 DATA 60,126,255,231,195,195,231,126, 60, 60,126,231,195,195,231,255,126, 60
1039 DATA 60,126,255,231,195,195, 3, 3, 59,123,251,251,195,195,231,255,126, 60
1040 DATA 206,223,223,219,219,219,219,219,219,219,219,219,219,219,219,223,223,206

```

PROGRAM B-13. UPRT9X26/DHX. Upright nine-pin alphabet—Epson, double-hex.

```

10 'PROGRAM B-13: "UPRT9X26/DHX" -- UPRIGHT 9-PIN ALPHABET (EPSON, DOUBLE-HEX)
15 'Copyright (c) 1985 by John Warner Davenport
20 'USING DOUBLE-HEX, A DECIMAL CODES AND DOUBLE DENSITY ON THE FX-80
30 DEFSTR A-Z

```

```

40 GOSUB 500      'EPSON DHX DEFS
50 PRINT "FINISHED DHX CODING"
60 GOSUB 1010     'ALPHABET CHARACTER STRINGS
70 PRINT "FINISHED ALPHABET CODING"
80 LPRINT CHR$(27)"a";      'RESET
100 LPRINT A;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R;S;T;U;V;W;X;Y;Z;N1;N2;N3;N4;N5;N6;
    N7;N8;N9;N0;NH
110 STOP
500 '***** EPSON DOUBLE-HEXADECIMAL DEFINITIONS *****
510 A0=CHR$(0):A1=CHR$(1):A2=CHR$(2):A3=CHR$(3):A4=CHR$(4):A5=CHR$(5):A6=CHR$(6):
    A7=CHR$(7):A8=CHR$(8):A9=CHR$(9):AA=CHR$(10):AB=CHR$(11):AC=CHR$(12):AD=CHR$(13):
    AE=CHR$(14):AF=CHR$(15)
520 AG=CHR$(16):AH=CHR$(17):AI=CHR$(18):AJ=CHR$(19):AK=CHR$(20):AL=CHR$(21):AM=CHR$(22):
    AN=CHR$(23):AO=CHR$(24):AP=CHR$(25):AQ=CHR$(26):AR=CHR$(27):AS=CHR$(28):AT=CHR$(29):
    AU=CHR$(30):AV=CHR$(31)
530 B0=CHR$(32):B1=CHR$(33):B2=CHR$(34):B3=CHR$(35):B4=CHR$(36):B5=CHR$(37):B6=CHR$(38):
    B7=CHR$(39):B8=CHR$(40):B9=CHR$(41):BA=CHR$(42):BB=CHR$(43):BC=CHR$(44):BD=CHR$(45):
    BE=CHR$(46):BF=CHR$(47)
540 BG=CHR$(48):BH=CHR$(49):BI=CHR$(50):BJ=CHR$(51):BK=CHR$(52):BL=CHR$(53):BM=CHR$(54):
    BN=CHR$(55):BO=CHR$(56):BP=CHR$(57):BQ=CHR$(58):BR=CHR$(59):BS=CHR$(60):BT=CHR$(61):
    BU=CHR$(62):BV=CHR$(63)
550 C0=CHR$(64):C1=CHR$(65):C2=CHR$(66):C3=CHR$(67):C4=CHR$(68):C5=CHR$(69):C6=CHR$(70):
    C7=CHR$(71):C8=CHR$(72):C9=CHR$(73):CA=CHR$(74):CB=CHR$(75):CC=CHR$(76):CD=CHR$(77):
    CE=CHR$(78):CF=CHR$(79)
560 CG=CHR$(80):CH=CHR$(81):CI=CHR$(82):CJ=CHR$(83):CK=CHR$(84):CL=CHR$(85):CM=CHR$(86):
    CN=CHR$(87):CO=CHR$(88):CP=CHR$(89):CQ=CHR$(90):CR=CHR$(91):CS=CHR$(92):CT=CHR$(93):
    CU=CHR$(94):CV=CHR$(95)
570 D0=CHR$(96):D1=CHR$(97):D2=CHR$(98):D3=CHR$(99):D4=CHR$(100):D5=CHR$(101):D6=CHR$(102):
    D7=CHR$(103):D8=CHR$(104):D9=CHR$(105):DA=CHR$(106):DB=CHR$(107):DC=CHR$(108):DD=CHR$(109):
    DE=CHR$(110):DF=CHR$(111)
580 DG=CHR$(112):DH=CHR$(113):DI=CHR$(114):DJ=CHR$(115):DK=CHR$(116):DL=CHR$(117):DM=CHR$(118):
    DN=CHR$(119):DO=CHR$(120):DP=CHR$(121):DQ=CHR$(122):DR=CHR$(123):DS=CHR$(124):DT=CHR$(125):
    DU=CHR$(126):DV=CHR$(127)
590 E0=CHR$(128):E1=CHR$(129):E2=CHR$(130):E3=CHR$(131):E4=CHR$(132):E5=CHR$(133):E6=CHR$(134):
    E7=CHR$(135):E8=CHR$(136):E9=CHR$(137):EA=CHR$(138):EB=CHR$(139):EC=CHR$(140):ED=CHR$(141):
    EE=CHR$(142):EF=CHR$(143)
600 EG=CHR$(144):EH=CHR$(145):EI=CHR$(146):EJ=CHR$(147):EK=CHR$(148):EL=CHR$(149):EM=CHR$(150):
    EN=CHR$(151):EO=CHR$(152):EP=CHR$(153):EQ=CHR$(154):ER=CHR$(155):ES=CHR$(156):ET=CHR$(157):
    EU=CHR$(158):EV=CHR$(159)
610 F0=CHR$(160):F1=CHR$(161):F2=CHR$(162):F3=CHR$(163):F4=CHR$(164):F5=CHR$(165):F6=CHR$(166):
    F7=CHR$(167):F8=CHR$(168):F9=CHR$(169):FA=CHR$(170):FB=CHR$(171):FC=CHR$(172):FD=CHR$(173):
    FE=CHR$(174):FF=CHR$(175)
620 FG=CHR$(176):FH=CHR$(177):FI=CHR$(178):FJ=CHR$(179):FK=CHR$(180):FL=CHR$(181):FM=CHR$(182):
    FW=CHR$(183):FO=CHR$(184):FP=CHR$(185):FQ=CHR$(186):FR=CHR$(187):FS=CHR$(188):FT=CHR$(189):
    FU=CHR$(190):FV=CHR$(191)
630 G0=CHR$(192):G1=CHR$(193):G2=CHR$(194):G3=CHR$(195):G4=CHR$(196):G5=CHR$(197):G6=CHR$(198):
    G7=CHR$(199):G8=CHR$(200):G9=CHR$(201):GA=CHR$(202):GB=CHR$(203):GC=CHR$(204):GD=CHR$(205):
    GE=CHR$(206):GF=CHR$(207)
640 GG=CHR$(208):GH=CHR$(209):GI=CHR$(210):GJ=CHR$(211):GK=CHR$(212):GL=CHR$(213):GM=CHR$(214):
    GN=CHR$(215):GO=CHR$(216):GP=CHR$(217):GQ=CHR$(218):GR=CHR$(219):GS=CHR$(220):GT=CHR$(221):
    GU=CHR$(222):GV=CHR$(223)
650 H0=CHR$(224):H1=CHR$(225):H2=CHR$(226):H3=CHR$(227):H4=CHR$(228):H5=CHR$(229):H6=CHR$(230):
    H7=CHR$(231):H8=CHR$(232):H9=CHR$(233):HA=CHR$(234):HB=CHR$(235):HC=CHR$(236):HD=CHR$(237):
    HE=CHR$(238):HF=CHR$(239)

```

```

660 HG=CHR$(240):HH=CHR$(241):HI=CHR$(242):HJ=CHR$(243):HK=CHR$(244):HL=CHR$(245):
HM=CHR$(246):HN=CHR$(247):HO=CHR$(248):HP=CHR$(249):HQ=CHR$(250):HR=CHR$(251):
HS=CHR$(252):HT=CHR$(253):HU=CHR$(254):HV=CHR$(255)
670 Q1=CHR$(0):Q2=Q1+Q1:Q3=Q2+Q1:Q4=Q3+Q1:Q5=Q4+Q1:Q6=Q5+Q1:Q7=Q6+Q1:Q8=Q7+Q1:Q9
=Q8+Q1:QA=Q9+Q1:QB=QA+QA:QC=QB+QA:QD=QC+QA:QE=QD+QA
680 S1=CHR$(32):S2=S1+S1:S3=S2+S1:S4=S3+S1:S5=S4+S1:S6=S5+S1:S7=S6+S1:S8=S7+S1:S9
=S8+S1:SA=S9+S1:SB=SA+SA:SC=SB+SA:SD=SC+SA:SE=SD+SA
690 BW=AR+CC:BY=AR+CB:EY=AR+C5:EX=AR+C6:DY=AR+C7:DX=AR+CB:ZY=EY+DY:ZX=EX+DX:IY=AR
+BK:IX=AR+BL:NY=AR+CG:NX=AR+CH:WY=AR+CJ:WX=AR+CK:BZ=AR+CQ:BX=AR+B1+A5
700 Z1=CHR$(255):Z2=Z1+Z1:Z3=Z2+Z1:Z4=Z3+Z2:Z5=Z4+Z1:Z6=Z5+Z1:Z7=Z6+Z1:Z8=Z7+Z1:
Z9=Z8+Z1:ZA=Z9+Z1:ZB=ZA+ZA:ZC=ZB+ZA:ZD=ZC+ZA:ZE=ZD+ZA:ZF=ZE+ZA:ZG=ZF+ZA:ZH=Z
G+ZA:ZH=ZG+ZA:ZI=ZH+ZA:ZJ=ZI+ZA
710 P1=CL+FA+CL+FA+CL+FA: P2=FA+CL+FA+CL+FA+CL
720 RETURN
1000 '***** DOUBLE-HEX CODE STRINGS FOR LETTERS & NUMBERS *****
1010 RL=AR+CU+A1+AF+A0: RN=AR+CU+A1+AD+A0 'GRAPHIC RESERVATIONS FOR 30, 26
1020 A=RL+A0+E0+A7+E0+AC+E0+AK+A0+B4+A0+C4+A0+E4+A0+C4+A0+B4+A0+AK+A0+AC+E0+A7+E
0+A0+E0+A0+A0+A0+A0
1030 B=RL+E0+E0+Z1+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+CL+Q
1+B2+STRING$(5,0)
1040 C=RL+AS+Q1+B2+Q1+C1+Q1+STRING$(16,128)+C1+Q1+D3+STRING$(5,0)
1050 D=RL+E0+E0+Z1+STRING$(17,128)+C1+Q1+B2+Q1+AS+STRING$(5,0)
1060 E=RL+E0+E0+Z1+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+ES+STRING$(11,128)+G1+E0+STRING$(4
,0)
1070 F=RL+E0+E0+Z1+E0+E8+E0+E8+Q1+E8+Q1+E8+Q1+ES+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q
1+G0+Q1+STRING$(4,0)
1080 G=RL+AS+Q1+B2+Q1+C1+Q1+STRING$(8,128)+E6+E0+E4+E0+E4+E0+E4+E0+C7+E0+D4+Q1+S
TRING$(4,0)
1090 H=RL+E0+E0+Z1+E0+E8+E0+AB+Q1+AB+Q1+AB+Q1+AB+Q1+AB+Q1+AB+Q1+AB+Q1+AB+Q1+AB+Q1+E
0+E0+E0+STRING$(4,0)
1100 I=RL+STRING$(10,0)+E0+E0+Z1+E0+E0+E0+STRING$(14,0)
1110 J=RL+A4+Q1+A6+Q1+A7+Q1+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+E1+Q1+HU+Q
1+E0+STRING$(5,0)
1120 K=RL+E0+E0+Z1+E0+E8+E0+AB+Q1+AB+Q1+AK+Q1+AK+Q1+B2+Q1+B2+Q1+C1+Q1+G1+E0+STRI
NG$(4,128)+STRING$(4,0)
1130 L=RL+E0+E0+Z1+E0+E0+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E
0+A1+E0+STRING$(4,0)
1140 M=RL+Q1+E0+Z1+E0+C0+E0+B0+Q1+AG+Q1+AB+Q1+A4+Q1+AB+Q1+AG+Q1+B0+Q1+C0+E0+Z1+E
0+Q1+E0+STRING$(4,0)
1150 N=RL+E0+E0+Z1+E0+E0+E0+C0+Q1+B0+Q1+AG+Q1+AB+Q1+A4+Q1+A2+Q1+A1+Q1+E0+E0+Z1+E
0+E0+E0+STRING$(4,0)
1160 O=RL+AS+Q1+B2+Q1+C1+Q1+STRING$(14,128)+C1+Q1+B2+Q1+AS+STRING$(5,0)
1170 P=RL+E0+E0+Z1+E0+E8+E0+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+CG+Q
1+B0+STRING$(5,0)
1180 Q=RL+A0+Q1+B4+Q1+C2+Q1+E1+Q1+E1+Q1+E3+Q1+E5+Q1+E5+Q1+E3+Q1+E1+Q1+C2+E0+B4+Q
1+A0+STRING$(5,0)
1190 R=RL+E0+E0+Z1+E0+E8+E0+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E8+Q1+E
0+B0+E0+STRING$(4,0)
1200 S=RL+B3+E0+CH+Q1+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+E8+E0+C5+Q
1+H2+STRING$(5,0)
1210 T=RL+G0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+E0+Z1+E0+E0+E0+E0+Q1+E0+Q1+E0+Q1+E0+Q
1+G0+STRING$(5,0)
1220 U=RL+E0+Q1+HU+Q1+E1+Q1+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+Q1+E0+E1+Q1+HU+Q
1+E0+STRING$(5,0)

```

```

1230 V=RL+E0+Q1+HG+Q1+EB+Q1+A4+Q1+A2+Q1+A1+Q1+Q1+E0+A1+Q1+A2+Q1+A4+Q1+EB+Q1+HG+Q
1+E0+STRING$(5,0)
1240 W=RL+E0+Q1+Z1+E0+E1+Q1+A2+Q1+A4+Q1+AB+Q1+AG+Q1+AB+Q1+A4+Q1+A2+Q1+E1+Q1+Z1+E
0+E0+STRING$(5,0)
1250 X=RL+STRING$(4,128)+G1+E0+C1+Q1+B2+Q1+AK+Q1+AB+Q1+AK+Q1+B2+Q1+C1+Q1+G1+E0+S
TRING$(4,128)+STRING$(4,0)
1260 Y=RL+E0+Q1+E0+Q1+G0+Q1+B0+Q1+AG+Q1+AB+E0+A7+E0+AB+E0+AG+Q1+B0+Q1+G0+Q1+E0+Q
1+E0+STRING$(5,0)
1270 Z=RL+G0+E0+E0+E0+E1+E0+E3+E0+E4+E0+EC+E0+EB+E0+E0+E0+FG+E0+H0+E0+G0+E0+E0+E
0+E1+E0+STRING$(4,0)
1280 N1=RN+STRING$(8,0)+C0+E0+Z1+E0+Q1+E0+STRING$(12,0)
1290 N2=RN+D1+E0+C1+E0+E2+E0+E2+E0+E4+E0+E4+E0+E4+E0+EB+E0+E0+E0+CG+E0+B1+E0+STR
ING$(4,0)
1300 N3=RN+G3+Q1+E1+Q1+STRING$(4,128)+EB+E0+EB+E0+EB+E0+E0+E0+FB+E0+G5+Q1+E2+STR
ING$(5,0)
1310 N4=RN+A2+Q1+A2+Q1+A6+Q1+AE+Q1+AQ+Q1+BI+Q1+D2+Q1+G2+E0+Z1+E0+A2+E0+A2+STRING
$(5,0)
1320 N5=RN+HJ+Q1+EH+Q1+EG+E0+EG+E0+EG+E0+EG+E0+EG+E0+EG+E0+EG+E0+E9+Q1+G6+STRING
$(5,0)
1330 N6=RN+BU+Q1+C9+Q1+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+C5+Q1+D2+STRING
$(5,0)
1340 N7=RN+G0+Q1+E0+Q1+E0+Q1+E0+Q1+E1+E0+E2+Q1+E4+Q1+EB+Q1+EG+Q1+F0+Q1+G0+STRING
$(5,0)
1350 N8=RN+B2+Q1+CL+Q1+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+CL+Q1+B2+STRING
$(5,0)
1360 N9=RN+B3+Q1+CH+Q1+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+EB+E0+C1+Q1+DU+STRING
$(5,0)
1370 N0=RN+BU+Q1+C1+Q1+STRING$(14,128)+C1+Q1+BU+STRING$(5,0)
1380 NH=RN+CB+Q1+H0+E0+A1+Q1+A2+Q1+A4+Q1+AB+Q1+AL+E0+B9+E0+CA+E0+EA+E0+A4+E0+STR
ING$(4,0)
1390 RETURN

```

PROGRAM B-14. PERP7X13/DAT. Perpendicular seven-by-thirteen bold full character set.

```

10 'PROGRAM B-14: "PERP7X13/DAT" -- PERPEND. BOLD 7 X 13 UPPER & LOWER (TRS-80)
1000 REM -- CODES FOR CAPITALS (1001-1026)
1001 DATA A,0,0,0,0,99,99,99,99,127,127,99,99,62,28,0
1002 DATA B,0,0,0,0,63,127,99,99,63,63,99,99,127,63,0
1003 DATA C,0,0,0,0,62,127,99,3,3,3,3,99,127,62,0
1004 DATA D,0,0,0,0,63,127,99,99,99,99,99,99,127,63,0
1005 DATA E,0,0,0,0,127,127,3,3,31,31,3,3,127,127,0
1006 DATA F,0,0,0,0,3,3,3,3,31,31,3,3,127,127,0
1007 DATA G,0,0,0,0,62,127,99,99,123,123,3,99,127,62,0
1008 DATA H,0,0,0,0,99,99,99,99,127,127,99,99,99,99,0
1009 DATA I,0,0,0,0,60,60,24,24,24,24,24,24,60,60,0
1010 DATA J,0,0,0,0,30,63,51,51,48,48,48,48,120,120,0
1011 DATA K,0,0,0,0,99,51,27,15,7,7,15,27,51,99,0
1012 DATA L,0,0,0,0,127,127,3,3,3,3,3,3,3,3,0
1013 DATA M,0,0,0,0,99,99,99,99,107,107,127,119,99,99,0
1014 DATA N,0,0,0,0,99,99,99,115,123,111,103,99,99,99,0
1015 DATA O,0,0,0,0,62,127,99,99,99,99,99,99,127,62,0
1016 DATA P,0,0,0,0,3,3,3,3,63,127,99,99,127,63,0

```

```

1017 DATA Q,0,0,0,0,16,62,63,123,99,99,99,99,127,62,0
1018 DATA R,0,0,0,0,99,99,99,99,63,63,99,99,127,63,0
1019 DATA S,0,0,0,0,62,127,99,96,126,63,3,99,127,62,0
1020 DATA T,0,0,0,0,24,24,24,24,24,24,24,24,126,126,0
1021 DATA U,0,0,0,0,62,127,99,99,99,99,99,99,99,99,0
1022 DATA V,0,0,0,0,8,28,54,54,54,54,99,99,99,99,0
1023 DATA W,0,0,0,0,99,99,119,127,107,107,99,99,99,99,0
1024 DATA X,0,0,0,0,99,99,119,62,28,28,62,119,99,99,0
1025 DATA Y,0,0,0,0,24,24,24,24,60,126,102,102,102,102,0
1026 DATA Z,0,0,0,0,127,127,3,6,12,24,48,96,127,127,0
1027 DATA 0,0,0,0,0,62,127,99,99,99,99,99,99,127,62,0
1028 DATA 1,0,0,0,0,30,30,12,12,12,12,12,14,12,8,0
1029 DATA 2,0,0,0,0,127,127,3,6,60,120,96,99,127,62,0
1030 DATA 3,0,0,0,0,62,127,99,96,60,60,96,99,127,62,0
1031 DATA 4,0,0,0,0,48,48,48,127,127,51,54,60,56,48,0
1032 DATA 5,0,0,0,0,62,127,99,96,96,63,31,3,127,127,0
1033 DATA 6,0,0,0,0,62,127,99,99,127,63,3,99,127,62,0
1034 DATA 7,0,0,0,0,6,6,6,12,24,48,99,99,127,127,0
1035 DATA 8,0,0,0,0,62,127,99,99,62,62,99,99,127,62,0
1036 DATA 9,0,0,0,0,62,127,99,96,126,127,99,99,127,62,0
1037 DATA a,0,0,0,0,63,63,51,63,48,63,30,0,0,0,0
1038 DATA b,0,0,0,0,31,63,51,51,51,63,31,3,3,3,0
1039 DATA c,0,0,0,0,30,63,51,3,51,63,30,0,0,0,0
1040 DATA d,0,0,0,0,62,63,51,51,51,63,62,48,48,48,0
1041 DATA e,0,0,0,0,30,63,3,63,51,63,30,0,0,0,0
1042 DATA f,0,0,0,0,6,6,6,6,15,15,6,54,62,28,0
1043 DATA g,0,30,63,48,48,62,63,51,51,51,63,30,0,0,0
1044 DATA h,0,0,0,0,51,51,51,51,63,31,3,3,3,3,0
1045 DATA i,0,0,0,0,30,30,12,12,14,12,0,12,12,0,0
1046 DATA j,0,14,31,27,24,24,24,28,24,16,0,24,24,0,0
1047 DATA k,0,0,0,0,51,27,15,7,7,15,27,51,3,3,0
1048 DATA l,0,0,0,0,30,30,12,12,12,12,12,12,14,14,0
1049 DATA m,0,0,0,0,107,107,107,107,107,127,54,0,0,0,0
1050 DATA n,0,0,0,0,51,51,51,51,51,63,31,0,0,0,0
1051 DATA o,0,0,0,0,30,63,51,51,51,63,30,0,0,0,0
1052 DATA p,0,3,3,3,31,63,51,51,51,63,31,0,0,0,0
1053 DATA q,0,48,48,48,62,63,51,51,51,63,62,0,0,0,0
1054 DATA r,0,0,0,0,3,3,3,3,23,63,27,0,0,0,0
1055 DATA s,0,0,0,0,30,63,48,28,3,63,30,0,0,0,0
1056 DATA t,0,0,0,0,28,62,54,6,6,31,31,6,6,6,0
1057 DATA u,0,0,0,0,62,63,51,51,51,51,51,0,0,0,0
1058 DATA v,0,0,0,0,8,28,62,62,51,51,51,0,0,0,0
1059 DATA w,0,0,0,0,54,127,107,107,107,107,107,0,0,0,0
1060 DATA x,0,0,0,0,51,51,30,12,30,51,51,0,0,0,0
1062 DATA y,0,6,15,24,28,62,55,51,51,51,51,0,0,0,0
1063 DATA z,0,0,0,0,63,63,6,12,24,63,63,0,0,0,0

```

Note: The upright five-by-seven alphabet (B-3) is also presented in string sequences in the SPELLUPS program (program 7-1) and the perpendicular seven-by-ten alphabet (B-9) is presented in that form in the WRDCRAFT (program 7-2) and WALLSTOK (program 8-9) listings.

APPENDIX C

Checklists for Converting TRS-80 Programs for IBM-Epson and Apple Printers

Checklist for Converting TRS-80 to IBM-Epson Programs

1. Differences in BASIC (assumes IBM-Epson printer driven by IBM PC)

a. Print command Instead of using LPRINT, use PRINT# preceded by OPEN "LPT1:" AS #1: WIDTH #1, 255, especially if trouble codes may be generated by mathematical functions, code conversions, etc.

b. RND function The IBM PC's random (RND) function differs from the TRS-80's in several ways (see the manual for the IBM). Seeding the random-number generator is done by using a negative expression in RND() instead of the RAN-
DOM command.

2. Differences in dot codes and pin configurations (see table 4-1)

a. Range of dot codes IBM-Epson printers have 256 dot codes ranging from 0 to 255; TRS-80 printers have 128 codes ranging from 128 to 255.

b. High-bit-up (IBM-Epson) vs. low-bit-up (TRS-80) IBM-Epson printers have eight printhead pins numbered (1,

2, 4, . . . 128) from the bottom instead of seven pins numbered (1, 2, 4, . . . 64) from the top. As a result, TRS-80 dot patterns are vertical mirror images of those printed by the bottom seven pins of IBM-Epson printers.

3. Exiting graphic mode Delete CHR\$(30) in any LPRINT statements of the TRS-80 program except when it is immediately preceded by CHR\$(27). Return to text mode is automatic with IBM-Epson printers as soon as the number of dot columns reserved for graphics is exhausted.

4. Entering graphic mode (graphic reservation command) Instead of CHR\$(18), send CHR\$(27)“K”CHR\$(N1)CHR\$(N2) to the printer, with N1 and N2 specifying the number of dot columns (N) reserved for graphic printing. If the number of columns needed is less than 256, N2 will always be 0. Above that, use the exact combination of N1 and N2 to specify a known value of N, or more generally, if N is variable, precede the enter-graphic-mode command by the conversions $N2 = \text{FIX}(N/256)$ and $N1 = \text{FIX}(N - 256 * N2)$.

5. Print density Instead of the TRS-80's CHR\$(27);CHR\$(19) for the standard 60 DPI, CHR\$(27);CHR\$(23) for the elite 72 DPI, or CHR\$(27);CHR\$(20) for the condensed 100 DPI, use “K” for single density (60 CPI) or “L” for double density (120 DPI) in the graphic reservation. On the Epson FX-80 and RX-80 there is a range of densities (60, 72, 80, 90, 120 and 240 DPI) having code numbers (n) which can be inserted in the graphic reservation command CHR\$(27)“*”CHR\$(n)CHR\$(N1)CHR\$(N2). The nearest Epson density to the TRS-80's condensed density is 90 DPI (720 dots per line, n=6).

6. Horizontal positioning IBM/Epson printers have no equivalent of the TRS-80 direct-positioning command CHR\$(27);CHR\$(16);CHR\$(N1);CHR\$(N2) in graphic mode and only an approximation of it in text mode (horizontal tabbing commands). For precise positioning at specific dot columns in graphics, IBM-Epson must use repeated-blank codes (such as STRING\$(120,0)) or a combination of horizontal tabbing and repeated blanks (see programs 3-3 and 3-4 and explanations in chapter 3).

7. Elongated printing Instead of CHR\$(27);CHR\$(14) for starting and CHR\$(27);CHR\$(15) for stopping elongated (expanded) printing on TRS-80 printers, IBM-Epson printers use CHR\$(14) or CHR\$(27)“W1” for starting and CHR\$(20) or CHR\$(27)“W0” for stopping. On TRS-80 printers the start and stop commands must be given in text mode, and they carry over to graphic-mode printing, making each dot pattern print twice; there is no such carryover with IBM-Epson printers—elongated printing can be done only in text mode.

8. Boldface printing With IBM-Epson printers there is also no carryover of bold printing from text to graphics mode, so TRS-80 control codes for starting (CHR\$(27);CHR\$(31)) and stopping (CHR\$(27);CHR\$(32)) boldface graphics printing must be deleted or (for text-mode bold printing) replaced by the codes for starting emphasized (CHR\$(27)“E”) or double-strike (CHR\$(27)“G”) and stopping emphasized (CHR\$(27)“F”) or double-strike (CHR\$(27)“H”), respectively.

9. Underlining (in text mode only) Replace the TRS-80 codes CHR\$(15) (for “on”) and CHR\$(14) (for “off”) by the IBM-Epson codes CHR\$(27)“–1” and CHR\$(27)“–0” respectively.

10. Line feeds and carriage returns While CHR\$(10) and CHR\$(13) are universal codes for an ordinary line feed and a carriage return, respectively, how they actually operate will depend on the setting of one or two DIP switches on the printer, the use of trailing semicolons in BASIC print commands, and whether the computer has been designed to intercept them and convert them on their way to the printer. TRS-80 computers usually convert CHR\$(13) to a combination carriage return and line feed, and the IBM PC converts CHR\$(10) to that combination (which is a major reason for using PRINT# instead of LPRINT in graphics with the IBM). In many cases (especially in text mode) the TRS-80 codes can be left unchanged in converting programs. In other cases, determine by trial and error whether changing the control codes themselves, the print punctuation, the DIP switches, or possibly even the operating system’s printer driver will remedy any difficulties. For converting special

graphic line feeds ($\frac{7}{32}$ inch, $\frac{1}{32}$ inch, $\frac{1}{64}$ inch) see the bottom of table 3.1. For converting codes for reverse line feeds, see the bottom of table 2-2 and related text (your printer may not have a reverse line feed).

11. Miscellaneous Be on the lookout for more minor TRS-80 control codes that need to be translated or deleted, including CHR\$(8);CHR\$(N) for backspacing, CHR\$(27);CHR\$(17) for proportional pitch, CHR\$(27);CHR\$(N) for proportional spacing, CHR\$(27);CHR\$(18) for the correspondence font, CHR\$(20) for entering and CHR\$(19) for exiting "word processing mode," CHR\$(27);CHR\$(16);CHR\$(0);CHR\$(0) for left margin set, CHR\$(28);CHR\$(N) for repeat printing (use tables 2-2 and 3-1).

Checklist for Converting TRS-80 to Apple Programs

1. Differences in BASIC (assumes Apple II DOS 3.3 and Applesoft BASIC driving Apple Imagewriter printer using Super Serial interface)

a. Print commands Applesoft has no LPRINT command; the equivalent is PRINT after output has been shifted from the computer's screen to the printer by PR#1 (standing alone in a previous line of the program). PRINT will operate like LPRINT in delivering control codes to the printer, but cannot be used to send graphic dot codes. For the latter, PEEK and POKE commands, must be substituted in order to use the full range of dot codes and prevent unwanted prints from leaking "phantom" codes. With the Apple IIe, for example, the printer's ready status must be examined by PEEK(49305) and the dot code (DC) sent by POKE 49304,DC. Use the form:

```
100 IF PEEK(49305) <> 16 THEN GOTO 100
110 POKE 49304,DC
```

b. IF-THEN-ELSE Applesoft has IF...THEN in its syntax but does not have ELSE. (This is why the PEEK and POKE commands cannot be in a single program line.) Whenever ELSE appears in the same line with IF...THEN in TRS-80 programs, what immediately follows ELSE must be placed in the next line below.

c. RND function Applesoft's random (RND) function differs from TRS-80's in several ways (see the Applesoft

manual). Seeding the random-number generator requires a negative expression in RND instead of the RANDOM command.

d. STRING\$() This function, heavily used in TRS-80 graphics programs for repeating dot codes or text characters, does not exist in Applesoft BASIC. In many cases, a FOR-NEXT loop for repeating will have to substituted for STRING\$(). Perhaps even more often, the Imagewriter's built-in repeat printing function—control codes CHR\$(27)“Rnnn” in text mode and CHR\$(27)“Vnnnn” in graphic mode—can be used in place of STRING\$() but in graphics it is less hazardous to use POKE (inside a FOR-NEXT loop) than a dot code in a statement such as **PRINT CHR\$(27)“Vnnnn”CHR\$(DC)**.

e. Remarks With Applesoft, program remarks can only be set apart from the program itself by REM, not with apostrophes. The latter will trigger a syntax-error interruption wherever they occur, so all remarks in a TRS-80 program preceded by apostrophes must be deleted, and the apostrophes themselves must be deleted or replaced by REM. (If used at the end of a program line, REM must be preceded by a colon.)

f. Other BASIC words Other TRS-80 BASIC words and phrases may have to be deleted where there is no Applesoft equivalent (CLEAR n, DEFINT, DEFSTR, PRINT@, PRINT USING, SPACE\$, VARPTR, WHILE...WEND, WIDTH) or replaced by Applesoft equivalents or approximations (HOME for CLS, INT for FIX, POKE for OUT, SCRN for POINT, PLOT for SET and RESET). In using STR\$() for converting numerical to string values, TRS-80 BASIC includes leading zeros in determining the length (LEN) of a string, whereas Applesoft does not add leading zeros in the first place.

2. Differences in dot codes and pin configurations (see table 4-2)

a. Range of dot codes Apple printers have 256 dot codes, ranging from 0 to 255; the TRS-80 has 128, ranging from 128 to 255. Any TRS-80 dot code can be converted to an Apple code by subtracting 128.

b. Eight-bit versus seven-bit coding Apple printers have eight pins numbered (1, 2, 4, . . . 128) from the top down, whereas TRS-80 printers have seven pins numbered (1, 2, 4, . . . 64) from the top down. Apple codes in the 0–127 range and TRS-80 codes in the 128–255 range print the same dot patterns using the top seven pins.

3. Exiting graphic mode Delete CHR\$(30) in any LPRINT statements of the TRS-80 program. Return to text mode is automatic with Apple printers as soon as the number of dot columns reserved for graphics is used up.

4. Entering graphic mode (graphic reservation command) Replace the TRS-80's CHR\$(18) with CHR\$(27) "Gnnnn" or CHR\$(27)"Snnnn" where nnnn, a string variable that specifies the number of dot columns reserved, can vary from 0001 to 9999 and may often have a fixed value that can be specified ahead of time. If not, define a numeric variable (e.g., RES=J*K) that represents the number of dot columns needed, and send a sequence of six bytes to the printer consisting of code 27, "G" or "S", the leading zeros (or blanks) in nnnn, and the remaining integer value of RES. In Microsoft BASIC this can be accomplished by CHR\$(27) + "G" + STRING\$(4-L, "0") + RIGHT\$(STR\$(RES), L) where L=LEN(STR\$(RES))-1, assuming a leading zero is added to the value of RES by BASIC. Since Applesoft has no STRING\$ function and no leading zeros in conversions by STR\$(), the third part of the sequence will have to be done with a FOR-NEXT loop and the entire sequence done as follows:

```
100 PR#1
110 RES=J*K: L=LEN(STR(RES))
120 PRINT CHR$(27)+"G";
130 FOR Q=L TO 4-L: PRINT "0";: NEXT Q
140 PRINT RIGHT$(STR$(RES),L);
```

5. Print density The Apple printers have seven print densities ranging from 72 to 160 DPI. Only one of these is identical to TRS-80 densities. This is the 72-DPI density, for which the TRS-80's CHR\$(27);CHR\$(23) should be replaced by CHR\$(27)"n". This is the closest you can get to the stan-

dard 60-DPI of the TRS-80 printers. The TRS-80's 100-DPI density is approximated by Apple's CHR\$(27)"E" (96 DPI).

6. Horizontal positioning The Apple equivalent of the TRS-80 direct-positioning command CHR\$(27);CHR\$(16);CHR\$(N1);CHR\$(N2) is CHR\$(27)"Fnnnn" where nnnn is a string variable (four ASCII numerals) ranging from 0000 to 1280. If the particular dot column targeted is a known constant, its number can be placed inside the quotation marks with leading zeros (e.g., "F0462"). If the positioning target is variable, use the sequence in lines 100–140 above, substituting the dot-column variable for RES and "F" for "G".

7. Elongated printing Substitute CHR\$(14) for the TRS-80's CHR\$(27);CHR\$(14) and CHR\$(15) for CHR\$(27);CHR\$(15) for starting and stopping elongated (expanded) printing, respectively.

8. Boldface printing Substitute CHR\$(27)"!" for the TRS-80's CHR\$(27);CHR\$(31) and CHR\$(27)"'"'"'" for CHR\$(27);CHR\$(32) for starting and stopping bold printing, respectively. As with TRS-80 printers, the bold feature carries over from text to graphic mode.

9. Underlining (in text mode only) Replace the TRS-80 codes CHR\$(15) for "on" and CHR\$(14) for "off" by Apple codes CHR\$(27)"X" and CHR\$(27)"Y" respectively.

10. Line feeds and carriage returns Note the comments about the interchangeability or automatic coupling of CHR\$(10) and CHR\$(13) in this category of the IBM-Epson checklist above. See table 2-2 for Apple's codes for 1/6-inch, 1/8-inch, and reverse line feeds (1/12-inch feeds are not available). Special graphic line feeds such as 7/72-inch, N/72-inch, and N/216-inch do not exist on Apple printers, but they can be approximated by codes for N/144-inch feeds, CHR\$(27)"Tnn" where nn has a range from 01 to 99. The appropriate substitution must be made wherever the TRS-80 code sequences CHR\$(27);CHR\$(50) for 1/72-inch and CHR\$(27);CHR\$(51) for 1/216-inch line feeds are found in the program to be converted.

11. Repeat printing Substitute CHR\$(27)“Vnnnn” where nnnn can range from 0000 to 9999, for the TRS-80’s CHR\$(28);CHR\$(N);CHR\$(DC) sequence for repeating a dot code. Many instances in which the STRING\$() function is used in TRS-80 programs can also be handled by the CHR\$(27)“Vnnnn” sequence. Use CHR\$(27)“Rnnn” for repeating a text character outside of a graphic reservation.

12. Miscellaneous Note Apple equivalents, or lack thereof, of TRS-80 control codes CHR\$(8);CHR\$(N) (for backspacing), CHR\$(27);CHR\$(17) (for proportional pitch), CHR\$(27);CHR\$(18) (for correspondence font), CHR\$(20) (for entering “word processing mode”), CHR\$(19) (for exiting that mode), CHR\$(27);CHR\$(N) (for proportional spacing), and CHR\$(27);CHR\$(16);CHR\$(0);CHR\$(0) (for left margin set) in tables 2-2 and 3-1.

Index

Index

- Accounting table program, 386–388
- ACCTABLE (accounting table) program, 386–88
- All-ranks table, 396–98
- Alphabet
 - arrays, 244–50
 - bold, 221–22
 - characters as data point symbols, line graphs, 320–322
 - coding sheets, 212–15
 - condensed, 218
 - near-normal-size, 215–22
 - nine-pin, 222–23
 - serif, 220
 - smallest, 215
- ALPHABET (for storage of code sequences) program, 428
- American League
 - games won (1977–1984), 407–408
 - runs scored and runs allowed (1901–1960), 273–75, 331
- Apple
 - checklist for converting TRS-80 programs to, 445–49
 - dot codes, 84–86
- Architectural drawing program, 375–79
- AREABARS (area graph using segmented bars) program, 297–300
- Area graph with bars, 297–300
- ARROW (simple bit-image form) program, 88–89
- Auto-theft and murder, rank-order correlation, 351–53
- B, data codes for Brush letter, 239–45
- Backspacing, printhead, 67–68
- Baenziger, Peter, 104, 106
- Bar-and-line graphs, tandem, 335–37
- Bar graph
 - area, 297–300
 - floating vertical, 286–96
 - horizontal, 260–76
 - means and standard errors, 279–86
 - segmented horizontal, 270–271
 - simple, 260–63
 - 3-D, 266–69
 - two-way horizontal, 271–76
 - vertical, 276–300

- BARGRAPH** (simple bar graph)
 program, 260–66
BARSIN3D (three-dimensional
 bar graph) program, 267–
 269
Baseball Graphics (Davenport),
 252, 382
Baseball's Pennant Races
 (Davenport), 252
BASIC
 commands for horizontal
 lines, 110
 commands for vertical lines,
 111
 detours around trouble
 codes, 78–79
 differences, TRS-80 vs.
 Apple, 445–46
 differences, TRS-80 vs. IBM-
 Epson, 442
 printing commands, 30–31
 typography in, 235–59
Basketball tournament diagram
 with line-graphic
 characters, 371–75
Batting averages
 ladder graph, 356–57
 line graph, 335–36
Bialzik, Jerry, 76–77
Binary dot codes, 80–98
Binary sum, 87
Bit image
 complex diagrams, 382–84
 graphics, 46–48
 screen dumps, 98–107
BLACKCAT program, 149–50
Black-white reversals, 101
 dot codes, 86, 87–89
Block and line graphic
 characters, diagrams with,
 368–80
Bold alphabet, 221–22
Boldface printing
 control codes, 36–37
 TRS-80 vs. Apple, 448
 TRS-80 vs. IBM-Epson, 444
Boothe, Bob, 76
Borders, 181–90
Boston Globe, 403
Boston Red Sox
 individual graphs of team's
 best hitters, 408–10
 offense, 269
 offense-defense graph, 272–
 273
 runs scored and allowed
 (1901–1983), 405–8
BRUSHB (data codes for Brush
 letter B) program, 239–45
Buffer, space and operation,
 control codes, 43–44
Cardinals. See St. Louis
 Cardinals
Carriage returns
 TRS-80 vs. Apple, 448
 TRS-80 vs. IBM-Epson, 444–
 445
Cell frequency, 355
CENTABLE (column-centering
 table) program, 388–92
Cervantes, J. Anthony, 36
Characters
 custom, control codes, 44
 dot-matrix printer, 27–30
Chicago White Sox
 decade display, 411, 414
 runs scored and runs allowed
 (1901–1960), 273–75, 331
Churchill, Winston, 171–72
Circular forms, 122–24
Coding sheets, numbers and
 letters, 212–15
Column-centering table
 program, 388–92
Complex math-generated
 figures, 130–33
Condensed alphabet, 218

- Control codes, 42–44
 - graphic-mode, 48–56
 - printer, 32–42
- Conversions
 - dot codes, 92–98
 - programs, 304
- Coordinates, polar, 124–30
- CUBETINT (drawing with shading levels) program, 161–69, 173
- Curved lines, 118–21
- Curves, and lines, 108–21
- Custom characters, control codes, 44
- DAGWOOD program, 144–45, 150, 177
- Data point symbols, line graphs, 312–15
 - alphabetic characters as, 320–322
- DBLHEXER (conversion of decimal to double-hex codes) program, 94–95
- DECBORDS (randomly selected dot patterns for borders) program, 183–84, 192, 193, 194, 199, 202
- DENNIS (cartoon drawing) program, 156, 160, 177, 380, 421–22
- Density, print, 443
 - control codes, 35–36
- Designs and forms, mathematical, 122–33
- Detours in BASIC, around trouble codes, 78–79
- Detroit Tigers, daily game log, 392–96
- DHEXPRT (assignment and printing of double-hex codes) program, 95–97
- Diagnostic aids, trouble codes, 72–75
- Diagrams with block and line graphic characters, 368–80
- DIP switches, control codes, 43
- Direct-positioning command, printhead, 58–60
- Dot codes
 - binary, 80–98
 - TRS-80 vs. Apple, 446–47
 - TRS-80 vs. IBM-Epson, 442–443
- Dot-matrix printers
 - compared to other printers, 22–25
 - three families of, 25–30
- Dot pattern, repeating, 52–54
- Dotplot-80 programs, 36
- DOTPRINT program, 47
- Dot-style typefaces, 255–59
- Double-hex codes, 94–97
- Drafting tapes, 179–80
- Drawing
 - simple line, 133–55
 - textured, 155–78
- DUMPSIZE (varying size of screen dumps) program, 103–5
- Electrosensitive printers, 23–24
- Elongated printing, 443–44
 - TRS-80 vs. Apple, 448
 - TRS-80 vs. IBM-Epson, 443–444
- ENLARGER program, 174–77
- Entering, graphic mode, 48–52
- EPSTOAPL (dot code conversion from Epson to Apple), 92–93, 150, 187
- Errors, standard, means with, 279–85
- Exiting, graphic mode, 48–52
- Figures, complex math-generated, 130–33

- FINALIST program, 411
- FINEDRAW (ultra-hi-res drawing) program, 205–11, 228, 238, 242–43
- Floating bar graph, vertical, 286–97
- Forms and designs, mathematical, 122–33
- FOTOGRAF program, 168–173, 176, 178, 381
- FREQDIST (frequency distribution) program, 276–279, 284, 285
- Frequency distributions, 276–279
- FUNCJAZZ (plotting concurrent math functions) program, 119–20, 121, 124
- GAMEGRAM/DAT (data for baseball game diagram) program, 383–84
- GAMELOG (Detroit Tigers' first 43 games of 1984) program, 392–96
- G clef symbol (FINEDRAW) program, 205–11
- GRABLE program, 396–98
- Graph(s)
 - range, 322–35
 - rough overviews, 400–402
 - and table combinations, 392
- Graphic characters, block and line, diagrams with, 368–380
- Graphic inserts, tables, 392–96
- Graphic mode
 - control codes, 48–56
 - entering and exiting, 48–52, 443, 447
- Graphics, future, 416–19
- Graphics characters, 28–30
- GRID7X7 (for making seven-by-seven coding sheets) program, 138, 144
- GRUMP (graphic screen dump) program, 104, 360
- HALOWEEN (jack-o'-lantern drawing) program, 134–42, 150
- Hard copy, microcomputers as producers of, 22–25
- Hex dump, 72–73
- Hinrichs, Delmer, 130, 132
- Histogram, frequency distribution, 276–79
- HOLLYWRD (Hollywood Lights typeface) program, 256–59
- Horizontal bar graphs, 260–76
- Horizontal lines, 109–10
- Horizontal positioning, 443
 - TRS-80 vs. Apple, 448
 - TRS-80 vs. IBM-Epson, 443
- Horizontal tabbing, 57
 - control codes, 40–42
- IBM-Epson
 - checklist for converting TRS-80 programs to, 442–45
 - dot codes, 80–84
- Illness and stress, scatterplot, 348–51
- Impact printer, 24–25
- Informal Gothic typeface, 252–255
- Ink-jet printers, 23–24
- Inserts, graphic, 392–96
- Jack-o'-lantern drawing, 134–142, 150
- Kalinowski, Francis, 84, 97–98, 173
- Karsh, Yousuf, 172–73
- Kater, David A., 76, 77, 129, 132, 211, 419
- Keller, Mike, 99–100, 160, 161, 419

- Ladder graphs, 356–60
- Lautzenheiser, Robert, 403
- Letters
 - coding sheets, 212–15
 - and numbers, 212–35
- Letter-segment selection, 252–255
- Lien, David, 76
- Line(s)
 - and curves, 108–21
 - oblique, 116–17
 - slanting, 115–17
 - and textures combined, 168–177
- Line and block graphic
 - characters, diagrams with, 368–80
- Line drawings, simple, 133–55
- LINEDRAW (universal line drawing) program, 150–56, 161, 168, 193, 205–11, 238, 242–43, 380, 381, 382
- Lined tables, 385–88
- Line feeds
 - control codes, 37–40
 - TRS-80 vs. Apple, 448
 - TRS-80 vs. IBM-Epson, 444–445
- LINEGRAF (plotting of line graphs with or without data points) program, 305–314
- Line graphs
 - alphabetic characters as data point symbols, 320–22
 - diagram, 371–80
 - multiple, 315–20
 - program, 305–14
- Line-matrix printer, 24
- Logos, 211
- Macworld*, 418
- MADHOUSE (architectural drawing) program, 375–79
- Main axis, horizontal bar graph, scaling, 263–66
- Manual polishing, graphics, 40–58
- Maps, 380–82, 423–27
- Mathematical forms and designs, 122–33
- Mathematical functions, plotting concurrent, 119–120, 121, 124
- Math-generated figures, complex, 130–33
- Means, with standard errors, bar graph, 279–86
- Microcomputer Graphics: Techniques and Applications* (Hearn and Baker), 106
- Microcomputers, as producers of hard copy, 22–25
- MINIPRNT subroutine (for miniature screen dump) program, 99–100
- MIRACLES (decorative borders and patterns) program, 185–90, 192, 193, 202
- Montages, 415–17
- MSEGRAPH (bar graph of means and standard errors) program, 279–286
- Multigraphs, 408–15
- Multiple-line graphs, 315–20
- MULTLING (multiple-line graph) program, 315–20
- Murder and auto theft, rank-order correlation, 351–53
- Mutual funds graph, 267
- National League
 - earned run averages, 401–2
 - season series summaries (1979), 415
- Near-normal-size alphabet, 215–222

- NETWORK program, 130
 Neuman, Jeffrey, 396
 Nine-pin alphabet, 222–23
 Numbers
 coding sheets, 212–15
 and letters, 212–35
 NUMS5X11/STR
 (perpendicular five-by-eleven numbers stored in strings) program, 432–33

 Oblique lines, 115–17
 OPTICART (Epson-to-Tandy conversion of SYMMETRY) program, 132–33

 PEARSONR (correlation scatterplot) program, 339, 342–51
 PENGUIN program, 137–45, 150
 PERP5X5/DAT (perpendicular five-by-five capitals and numbers) program, 429
 PERP5X7/DAT (perpendicular five-by-seven capital letters) program, 430–31
 PERP5X11/DAT (perpendicular five-by-eleven condensed characters) program, 431–32
 PERP7X10/DAT
 (perpendicular bold seven-by-ten capitals and numbers) program, 434–35
 PERP7X13/DAT
 (perpendicular seven-by-thirteen bold full character set) program, 440–41
 PERP716S/DAT (perpendicular seven-by-sixteen serif capitals) program, 436–37
 PERP8X18/DAT
 (perpendicular eight-by-eighteen bold capitals and numbers) program, 437
 Perpendiculars, spelling with, 229–35
 PERPSERF/DAT
 (perpendicular seven-by-eleven serif capitals and numbers) program, 435–36
 Pictograms, 300–302
 true, 302–4
 PICTOGRM (pictogram with partial figures) program, 301–3, 381
 Pictures, drawing, 133–78
 PIECHART (exploded pie chart) program, 361–66, 415
 Pie charts, 360–67
 Pin configurations
 TRS-80 vs. Apple, 446–47
 TRS-80 vs. IBM-Epson, 442–443
 Pitch. *See* Print density
 Pixel, 99
 Plotting, precision, 402–4
 PNTRANGE (range, point-and-range, and rank-and-range graphs) program, 322–37, 396, 397, 400, 411
 Point-and-range graphs, 322–337
 Polar coordinates, 124–30
 POLARRAY program, 126–30, 361
 POSITION (direct-positioning) program, 59–60, 66
 Positioning, horizontal, 443
 Precision plotting, 402–4
 Print buffer, space and operation, control codes, 43–44
 Print density, 443
 control codes, 35–36
 TRS-80 vs. Apple, 447–48
 TRS-80 vs. IBM-Epson, 443

- Printer
 - control codes, 32–42
 - graphics, future, 416–19
 - “Printer as a Paintbrush, The” (Keller), 160
- Printhead, positioning, 56–68
- Printing
 - BASIC commands, 30–31
 - boldface, 36–37, 444, 448
 - elongated, 443–44
 - samples, 28–29
 - speed control codes, 43
- PRNTDRVR/SUB (Boothe’s printer driver) program, 76
- Proportional spacing command, printhead, 67
- RABBITS program, 144–45, 150
- Range graph program, 322–37
- Rank-and-range graph program, 322–37
- Rank correlations, 351–56
- RANKPLOT (rank-order correlation scatterplot) program, 351–56
- Red Sox. *See* Boston Red Sox
- Remedies, trouble codes, 75–78
- REPBLANK (repeat-blank positioning) program, 64–66
- Repeat-blank positioning, printhead, 61–66
- Repeating, dot pattern, 52–54
- Repeat printing, TRS-80 vs. Apple, 449
- Reverse line feeds, control codes, 39–40
- St. Louis Cardinals, time-series graph, 396–97
- Scaling, main axis, horizontal bar graph, 263–66
- Scatterplots, 338–56
- Screen
 - bit-image dumps, 98–107
 - tints, textures, and, 190–204
- SEGBARS (segmented horizontal bar graph) program, 270–71
- Segmented horizontal bar graph, 270–71
- Serif alphabet, 220
- SHADINGS (15 different densities) program, 194–204
- Simple bar graph, 260–63
- Slanting lines, 115–17
- SLOPES (oblique lines) program, 116–17
- Sloping lines, 115–17
- Smallest alphabet, 215
- Smith, Alan D., 129
- Speed, printing, control codes, 43
- Spelling
 - with perpendiculars, 229–35
 - upright-oriented words, 226–228
- SPELLUPS (spelling of upright-oriented words) program, 225–28
- Spotlighting, 156–60
- SPOTLITE (subroutines for shading levels) program, 157–60, 194
- Standard errors, means with, 279–85
- Statistical tapes, 180–81
- Stock, price-and-volume plot, 290–97
- Stock market, and Super Bowls, 338–48
- STRAUS (letter segments) program, 252–55
- Straus Printing Co., 252
- Stress and illness, scatterplot, 348–51

- Super Bowls, and stock market, 338–48
- SYMBLGRF (symbol graph)
 - program, 300–301
- Symbol graph, 301–2
- Symbols, 204–11
 - line graph data points, 312
- SYMMETRY program, 132

- Tabbing, horizontal, control
 - codes, 40–42
- TABBYCAT program, 149–50
- Table-and-graph combinations, 392
- Tables, 384–85
 - all-ranks, 396–98
 - column-centered, 388–92
 - and graph combinations, 392
 - graphic inserts, 392–96
 - lined, 385–88
- TABMATH1 (sine curve
 - drawing) program, 41
- TABMATH2 (sine and cosine
 - curve) program, 42
- Tandem bar-and-line graphs, 335–37
- Tapes
 - drafting, 179–80
 - statistical, 180–81
- TEMPBARS (floating bar graph
 - of daily temperature
 - ranges) program, 286, 287–290, 291
- Temperature, floating bar graph
 - of daily ranges, 287–90
- Textured drawings, 155–68
- Textures, screen, and tints, 190–204
- Thermal printers, 23
- Thomas, Susan J., 129
- 3-D bar graph, 266–69
- Tigers. *See* Detroit Tigers
- Tints, screen, and textures, 190–204
- TOTMPOLE (ladder graph)
 - program, 356–60
- TRISWEEP (three-pass
 - homemade letters)
 - program, 235–38
- TRNYGRAM (tournament
 - diagram with line-graphic
 - characters) program, 371–375
- Trouble codes, 68–79
- TRS-80
 - checklist for converting to
 - Apple programs, 445–49
 - checklist for converting to
 - IBM-Epson programs, 442–445
 - dot codes, 86–87
- TRSTOEPS (mirror-image
 - conversion of 78-bit dot
 - codes) program, 92–93, 150, 187
- True pictograms, 302–4
- Two-way bar graphs, 271–76
- TWOWAYBG (two-way
 - horizontal bar graph)
 - program, 272–74, 392
- TYPARRAY (alphabetic array)
 - program, 246–51, 303
- Type, dot-matrix printer, 27–30
- Typefaces
 - dot style, 255–59
 - Hollywood Lights, 256–59
 - Informal Gothic, 252–55
- Typography, in BASIC, 235–59

- ULTHIRES (ultra-high-
 - resolution vertical lines)
 - program, 112–14, 202
- Ultra-high resolution
 - symbols, 205–11
 - vertical lines with, 112–15
- Underlining
 - TRS-80 vs. Apple, 448
 - TRS-80 vs. IBM-Epson, 444

- UPDN5X7/DAT (upside-down five-by-seven capital letters) program, 431
- Upright words, spelling program, 227–28
- UPRT5X7/DAT (upright five-by-seven capital letters) program, 429–30
- UIPRT9X26/DHX (upright 9-pin alphabet, double-hex) program, 437–40
- UPRTITAL/DAT (upright italic full character set) program, 433–34
- USAMAP (map of U.S.) program, 380, 381, 423–27
- Vertical bar graphs, 276–300
- Vertical floating-bar graphs, 286–96
- Vertical lines, 110–12
 - with ultra-high resolution, 112–15
- WALLSTOK (price-and-volume stock plot) program, 290–297
- Weather log, 391
- WHITECAT program, 146–48, 150, 151, 208, 238, 382
- WHITEDOG program, 150
- White Sox. *See* Chicago White Sox
- Whole-character programming, 238–44
- Williams, Ted, batting average line graphs, 335–36
- Word-spelling programs, 223–235
- World Series of 1982, 410–13
- WRDCRAFT (spelling for perpendiculars) program, 230–35, 297, 319, 403
- XYCIRCLE (drawing circles or ellipses) program, 122–23

Your Dot-Matrix Printer Can:

- produce letter-quality documents
- custom-design type fonts
- create high-quality graphic images, borders, and much more!

If you are tired of watching a sharp-looking graphic on your computer screen turn into a heap of jagged lines and inaccurately aligned graphs when you print it out, *Graphics for the Dot-Matrix Printer* is the book for you—a definitive guide to programming your printer to create business graphs and charts, custom type fonts, pie charts, maps, mathematical shapes, artwork, decorative elements such as borders and trims . . . indeed, all kinds of high-resolution graphics.

With *Graphics for the Dot-Matrix Printer* you can relieve the frustration of dealing with graphics software that creates printed images by “dumping” them directly from the computer screen. This method doesn’t take advantage of the fact that the printer has a much higher resolution capacity than a video monitor. With John Warner Davenport’s clear explanations and easy-to-follow BASIC routines for all of the major brands of dot-matrix printers available for the IBM PC and all Apple and Radio Shack computers, you’ll learn how to generate crisp, realistic, and accurate images you could not get otherwise. With over 150 computer-generated graphics created by the author, this book is an invaluable and one-of-a-kind companion for performing miracles you’ll never see on your computer screen.

**Computer Book Division
Simon & Schuster, Inc.**

Cover design ©1985 by Hal Siegel